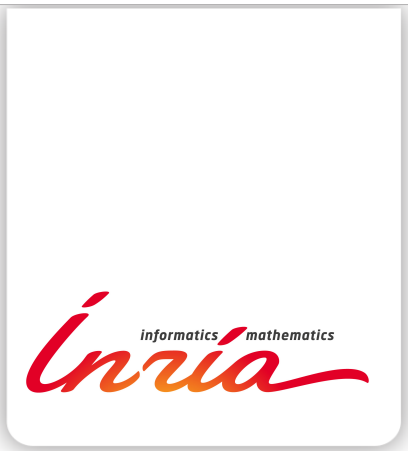
Implementation of three types of ordinals in Coq

José Grimm

RESEARCH REPORT
N° 8407
November 2013
Project-Team Marelle

ISSN 0249-6399 ISRN INRIA/RR--8407--FR+ENG



Implementation of three types of ordinals in Coq

José Grimm*

Project-Team Marelle

Research Report n° 8407 — November 2013 — 74 pages

Abstract: One can define an inductive type T in Coq by the rules: zero is in T, and 'cons a n b' is in T when a, b are in T and n is an integer. One can embed this type with an ordering, and show that the subset of "normal" elements is well-ordered, thus corresponds to some ordinal, namely epsilon-zero, the least epsilon-ordinal of Cantor. The same construction can be applied to the case where cons take one more argument of type T; in this case we get the Feferman-Schütte ordinal Gamma-zero. These two type were implemented by Castéran in Coq. In these paper we present an implementation using the ssreflect library. One can consider the case where cons takes four arguments of type T and use the ordering function proposed by Ackermann. This gives some large ordinal. The proof in Coq that it is a well-ordering matches exactly that of Ackermann. Every limit ordinal in this type is (constructively) the supremum of a strictly increasing function. Finally, we show how these types are related to ordinals defined in an implementation of the theory of sets of Bourbaki (a variant of Zermelo-Fraenkel).
The code is available on the Web, under http://www-sop.inria.fr/marelle/gaia.

Key-words: Gaia, Coq, Bourbaki, ordinals, well-order

* Email: Jose.Grimm@inria.fr

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Trois types d'ordinaux dans Coq

Résumé : On peut définir dans Coq un type T par les deux règles: zero est de type T, et 'cons a n b' est de type T si a, b sont de type T et n est entier. On peut munir ce type d'une relation d'ordre, et l'ensemble des éléments normaux est bien ordonné. Il correspond à un certain ordinal, dénoté epsilon-zéro par Cantor. En rajoutant un argument à la fonction cons, on obtient les ordinaux plus petits que Gamma-zéro, l'ordinal de Feferman-Schütte. Ces deux types ont été implémentés par Castéran dans Coq, nous l'avons porté dans ssreflect. En rajoutant un quatrième argument à la fonction cons, et en utilisant un ordre proposé par Ackermann, on obtient un grand ordinal. La preuve Coq que la relation est une relation d'ordre bien fondée reflète parfaitement les arguments de Ackermann. Tout ordinal limite de ce type est (de façon constructive) le sup d'une fonction strictement croissante. Enfin, nous faisons le lien avec ordinaux de la théorie des ensembles de Bourbaki (une variante de Zermelo-Fraenkel).

Le code est disponible sur le site Web http://www-sop.inria.fr/marelle/gaia.

Mots-clés : Gaia, Coq, Bourbaki, ordinaux, bon-ordre

Chapter 1

Introduction

In the report [6], we have described an implementation of ordinals numbers in COQ, using classical axioms of the Theory of Sets (in particular, the Excluded Middle Axiom and the Axiom of Choice). A similar work has been done by Pierre Castéran [1]. We have taken the two files epsilon0.v and gamma0.v, converted them into SSREFLECT syntax, removed every dependance on rpo (recursive path ordering), and filled the holes in some proofs. On the other hand, we have taken a paper by Ackermann, and implemented it in a similar fashion. In these three cases, we have a well-ordered countable set, that can be identified to some ordinal. In the last chapter, we show that the first two types correspond to the ordinals ϵ_0 and Γ_0 defined in [6].

On the the Web, under http://www-sop.inria.fr/marelle/gaia, one can find the code describe in this report. The code described in chapters 2, 3 and 4 is in the file *ssete9.v*. The definition of Γ_0 is in *sset14.v* (this files depends on many other files). Chapter 5 describes the link between *ssete9.v* and *sset14.v*; the corresponding code is in *ssete10.v*.

1.1 Properties of ordinals

We describe here briefly some properties of von Neumann ordinals; for simplicity, we shall just write "ordinal" in this chapter; however, in the next three chapters, an unqualified ordinal will be an object of one of the three types T_1, T_2 and T_3. An ordinal is a set x such that the relation $a = b$ or $a \in b$ between elements of x makes it a well-ordered set. For every well-ordered set $(E, \leq)$, there is a unique order isomorphism $f : E \to x$. This means that $a < b$ is equivalent to $f(a) \in f(b)$. The relation $x \subset y$ is a well-ordering between ordinals; we shall write it $x \leq y$. Each ordinal x is equal to the set of ordinals y such that $y < x$. No set contains all ordinals.

Let's define a *map* as a triple $F = (A, B, G)$ of sets, where $G \subset A \times B$ is a functional graph with domain A; if $x \in A$, the unique y such that $(x, y) \in G$ is denoted by $F(x)$ and called the value of F at x. One often identifies a function and its associated map (for instance addition of integers). Since no set contains all ordinals, there is no map associated to ordinal addition. One can define a function by transfinite induction by consider a bound, define a map on the set of ordinals less than this bound, and show that the result is independent of the bound.

One can define the successor x^+ of an ordinal, and by transfinite induction the functions $x + y$, $x \cdot y$ and x^y (there are alternate ways of defining addition and multiplication).

[1] see: http://www.labri.fr/perso/casteran, file Kantor.tar.gz

Let E be a set of ordinals. There is an ordinal x greater than all the elements of E (no sets contains all ordinals). There is a set containing all ordinals y, such that $y \leq x$ and $a \leq y$ whenever $a \in$ E; this set has a smallest element, called the *supremum* of E. Let x be an ordinal, and E the set of ordinals $< x$. If $x = 0$ this set is empty. Otherwise, if y is the supremum of E, it follows $y \leq x$. In the case $y = x$, the ordinal x is said to be a *limit* ordinal; otherwise x is the successor of y. The least limit ordinal is denoted ω. The set of natural integers **N** can be identified with the set of ordinals $< \omega$.

Let **C** be a regular uncountable cardinal (see [6] for a definition), and E the set of ordinals $<$ **C**. It happens that E is stable by addition, multiplication, exponentiation, ϕ_0, ψ, etc. Moreover, any countable subset of E has a supremum in E. If $\mathbf{C} = \aleph_1$, then E is the set of countable ordinals. Cantor [3, §15; Theorem K] says that every limit ordinal of E is the supremum of a strictly increasing family $(x_i)_{i\in\mathbf{N}}$.

We say that a function f is *normal* if it satisfies the two conditions

$$x < y \implies f(x) < f(y) \tag{1.1}$$

$$f(x) = \sup_{y<x} f(y), \quad x \text{ limit.} \tag{1.2}$$

If the second condition holds, then f is said to be *continuous* at x. If these two conditions hold, then $f(\sup \mathrm{E}) = \sup_{y\in\mathrm{E}} f(y)$ for any set of ordinals E. Assume that f and g are two normal functions, $f(0) = g(0)$, and there is a function P such that $f(x^+) = \mathrm{P}(f(x))$, and $g(x^+) = \mathrm{P}(g(x))$. Then $f = g$. Conversely, one can define, by transfinite induction, a function f such that $f(0)$ is given and $f(x^+) = \mathrm{P}(f(x))$. This function is normal (thus unique) provided that P is increasing and $\mathrm{P}(x) > x$.

Thus, by transfinite induction on x, one can define, for fixed a, three normal functions $a + x$, $a \cdot x$ and a^x. These operations coincide with the usual operations on **N**. However, one has $1 + \omega = \omega < \omega + 1$, so that many properties of integers do not hold (In the case of T_1 we shall list all valid properties or give counter-examples). These operations have an inverse: subtraction, division and logarithm. For instance, if $a \geq 2$ and $b \neq 0$, there exists x, y and z such that

$$b = a^x \cdot y + z, \quad z < a^x, 0 < y < a. \tag{1.3}$$

If we take for instance $a = \omega$, the condition $0 < y < a$ says that y is a non-zero integer. We thus can write

$$b = \omega^x \cdot (n+1) + z, \quad z < \omega^x. \tag{1.4}$$

If z is non-zero, one can iterate, and one gets

$$b = \omega^{x_1} + \omega^{x_2} + \ldots + \omega^{x_n}$$

where the sequence of x_i is decreasing. This is one way to express the Cantor normal form. One can also write

$$b = \omega^{x_1} \cdot n_1 + \omega^{x_2} \cdot n_2 + \ldots + \omega^{x_{n_k}} \cdot n_k$$

where the n_i are non-zero integers and the x_i strictly decreasing. These two form are completely equivalent to (1.4). If x is non-zero, one can also iterate on x, but there is no guarantee that the process terminates; this will be explained in a moment.

Let's consider a collection of ordinals (i.e., we consider a predicate p, such that if $p(x)$ holds, then x is an ordinal; if E is a set of ordinals, the predicate could be $x \in$ E). For any ordinal α, we consider E_α, the set of elements of the collection that are $< \alpha$. This is a well-ordered

set, thus isomorphic to some ordinal β. This means that there is a function f_α, defined for $x < \beta$, strictly increasing, whose range is E_α. We say that f_α *enumerates* E_α. If the collection is unbounded, there is a function f, defined for all ordinals, that coincides with f_α (for $x < \beta$) for every α. We say that f enumerates the collection. If $\mathbf{C}$ is a cardinal as above, $\alpha = \mathbf{C}$ and if E_α has cardinal $\mathbf{C}$, then $\beta = \alpha$.

For any normal function f (for instance $f(x) = a + x$ or $f(x) = a^x$), there are infinitely many fix-points. For instance, assume $1 + x = x$. By associativity, $1 + (x + y) = (1 + x) + y = x + y$. Thus, if $f(x) = 1 + x$, all infinite ordinals are fix-points. The function $x \mapsto \omega + x$ enumerates the fix-points of $f(x) = 1 + x$. The function that enumerates the fix-points of f is called its *first derived* function.

One says that a set E of ordinals is *closed* if the supremum of a family of elements of E is either the supremum of E or an element of E. A collection is closed if the supremum of a subset of the collection is in the collection (for instance, the collection of limit ordinals is closed; the set of countable limit ordinals is also closed).

We have ([8]): if f is a normal function, then the collection of its fix-points is closed; the enumeration function of a closed collection is a normal function. This means that the first derived function of a normal function is normal. Moreover, if $\mathbf{C}$ is as above, if f maps $E_{\mathbf{C}}$ to itself, then the first derived function is defined on $E_{\mathbf{C}}$.

Let f_i be a collection of functions, indexed by an ordinal $i \in I$. We assume that if $i < j$, every fix-point of f_j is also a fix-point of f_i. Thus, the set of common fix-points of all f_j (for $j \leq i$) is the set of fix-points of f_i. If each f_i is normal, and I not too big, then the set of all common fix-points of f_i is infinite and closed. In particular, one can take for I the set of ordinals $< k$ (where k is non-zero). This means that there exists an enumeration function f_k. Thus (see[8, Theorem 6]), one can define by transfinite induction, a family of functions f_i starting from f_0. If f_0 maps $E_{\mathbf{C}}$ onto itself, so does f_i for every i in $E_{\mathbf{C}}$.

We shall consider the family ϕ defined by $\phi_0(x) = \omega^x$, and take for $E_{\mathbf{C}}$ the set of countable ordinals. A number of the form $x = \omega^a$ is called an "additive principal number", or sometimes "additively indecomposable". It satisfies the property that $b + x = x$ whenever $b < x$. It satisfies also $a + b < x$ if $a < x$ and $b < x$. Thus ϕ_0 is the enumeration function of the collection of additive principal numbers, and can be defined without defining multiplication or exponentiation. If $\omega^x = x$, then x is called an epsilon-number by Cantor. The function ϵ_x enumerates all the epsilon-numbers; it is the first derived function of ϕ_0, thus is ϕ_1. To make the definition of ϕ a bit more precise, we introduce $C(\alpha)$, the set of all countable x, such that $\phi_\gamma(x) = x$, whenever $\gamma < \alpha$; then $\beta \mapsto \phi_\alpha(\beta)$ enumerates the set $C(\alpha)$.

We say that α is *strongly critical* if $\alpha \in C(\alpha)$, or if $\phi_\alpha(0) = \alpha$. There are many such numbers, the least of them is called Γ_0 (the so-called Feferman-Schütte ordinal). We consider now $\psi_\alpha(\beta)$ to be $\phi_\alpha(\beta')$ for some β' (see [7] for details). We lose the property that the function is normal, but gain in injectivity. The important property is that any $\phi_0(x)$ is uniquely of the form $\psi_a(b)$. We may rewrite (1.4) as

$$x = \phi_0(a) \cdot (n+1) + r, \quad r < \phi_0(a),$$

$$x = \psi_a(b) \cdot (n+1) + r, \quad r < \psi_a(b),$$

$$x = \Phi(a,b,c) \cdot (n+1) + r, \quad r < \Phi(a,b,c).$$

The last equation is proposed by Ackermann in [1]. It depends on a mysterious function Φ.

Let's define a type T_1 as follows. An object x is of type T_1, if it is either zero, or comes from two objects a and r of type T_1, and an integer n, we write it $[a, n, r]$.

Let's associate to each ordinal x an object $f(x)$ of type T_1 as follows. If x is zero, we associate zero. Otherwise, we use (1.4), this gives a, n and r. We define $f(x) = [f(a), n, f(r)]$. Note that f is not surjective, since not all elements of T_1 satisfy the equivalent of $r < \phi_0(a)$. On the other hand, if $x = \epsilon_y$, then $x = \phi_0(x)$, so that $f(x) = [f(x), 0, f(0)]$. Such an equation has no solution in T_1. For this reason we consider only numbers less than ϵ_0. We consider also type T_2 and T_3, using $\phi_a(b)$ or $\Psi(a,b,c)$.

Consider now the inverse function g of f. In the case of T_1, we get $g([a,n,b] = \phi_0(g(a)) \cdot (n+1) + g(b)$. As we assume that g is strictly increasing, as well as ϕ_0, this induces an ordering on T_1, that does not depend on the exact value of ϕ_0. Similarly, monotonicity of ψ gives an ordering on T_2. Finally, we may order T_3 by generalizing the ordering on T_2. Once we have an ordering on T_1, we can express the condition $r < \phi_0$, thus specify the domain of g. This can also be done in the other cases. Let's denote these domains by T_1', T_2', and T_3'. We can define g by transfinite induction: for every x, $g(x)$ is the supremum of all $g(y)$ for $y < x$. The difficulty here is to define a type T big enough so that it contains the image of g, and implement ordinals in T. We could for instance take T_2 when considering T_1'. Let's assume that such a T exists, so that, for any reasonable g, we can define $S(g,x)$, the supremum of all $g(y)$ for $y < x$. We have to construct g such that $g(x) = S(g,x)$, via transfinite induction. This is only possible if the ordering satisfies some condition, namely, that it is well-founded. Let α be the supremum of all $g(x)$. Every ordinal less than α is in the image of g. We say that α is the ordinal of our type (T_1, T_2 or T_3).

In this paper, we shall define the three types, and a comparison function. We shall show that the comparison function is an ordering. We shall define NF ordinals, and show that the comparison function is well-founded for NF ordinals. One could then deduce that each of these types has a ordinal. It is ϵ_0 for T_1, Γ_0 for T_2 and an unknown quantity for T_3. We shall not prove these facts.

We shall show how addition, multiplication and exponentiation can be defined on T_1. In the case T_2 and T_3, we define only addition. An interesting point is that Ackermann constructs explicitly a function $f_x(n)$, such that, for any limit ordinal x of the family, f_x is a function defined for all integers n, is strictly increasing and the supremum of the $f_x(n)$ is x.

1.2 Introduction to Coq

We assume the reader familiar with COQ. We indicate here some facts that may be useful. Our work is based on the SSREFLECT library, see [4]. In particular the script starts with

```
Require Import ssreflect ssrfun ssrbool eqtype ssrnat.
```

The set of natural integers is defined by the following line

```
Inductive nat : Set :=  0 : nat | S : nat -> nat.
```

From this, COQ generates the following induction principle.

```
nat_ind
     : forall P : nat -> Prop,
       P 0 -> (forall n : nat, P n -> P n.+1) -> forall n : nat, P n
```

In order to show $P(n)$ for every n it suffices to show '`P 0`' and that '`P n`' implies '`P (S n)`'. Note that 0 and 1 are standard notations for '`0`' and '`S 0`' while '`n.+1`' is the SSREFLECT notation for '`S n`'.

One can define a function f by induction, by specifying $f(0)$ and how to compute $f(n+1)$ given n and $f(n)$. Here is an example.

```
Fixpoint fact (n:nat) : nat :=
  match n with
    | O => 1
    | S n => S n * fact n
  end.
```

The SSREFLECT library provides the following definition. Note how the `match` construct has been replaced by an '`if _ is _ then_ else _`' construct.

```
Fixpoint fact_rec n := if n is n'.+1 then n * fact_rec n' else 1.
```

We show here how $p+n=p+m \implies n=m$ and $p+n \leq p+m \iff n \leq m$ are expressed in the standard library and SSREFLECT.

```
Lemma plus_reg_l : forall n m p, p + n = p + m -> n = m.
Lemma plus_le_compat_l : forall n m p, n <= m -> p + n <= p + m.
Lemma plus_le_reg_l : forall n m p, p + n <= p + m -> n <= m.

Lemma eqn_add2l p m n : (p + m == p + n) = (m == n).
Lemma leq_add2l p m n : (p + m <= p + n) = (m <= n).
```

There is an important difference between these two cases: the standard library provides an inductive function `le` whose type is `Prop`, with two constructors that assert $n \leq n$, and $n \leq m \implies n \leq m+1$, while SSREFLECT defines a boolean function `leq` such that $n \leq m$ is $n - m \equiv 0$; the equivalent of the two constructors are lemmas `leqnn` and `leqW`. Here $x \equiv y$ is the boolean comparison of two numbers; it is true if $x = y$, and false otherwise.

Whether $x \leq y$ is true or not is decidable; this is obvious in the boolean case. So given, two numbers n and m, one of $m < n$, $n < m$ or $m = n$ holds. This is asserted by the following definition and lemma.

```
CoInductive compare_nat m n : bool -> bool -> bool -> Set :=
  | CompareNatLt of m < n : compare_nat m n true false false
  | CompareNatGt of m > n : compare_nat m n false true false
  | CompareNatEq of m = n : compare_nat m n false false true.
Lemma ltngtP m n : compare_nat m n (m < n) (n < m) (m == n).
```

Since $m < n$ is decidable, it allows us to define functions by cases. Compare the definition of the maximum in the standard library (using fixpoint) and in SSREFLECT.

```
Fixpoint max n m : nat :=
  match n, m with
    | O, _ => m
    | S n', O => n
    | S n', S m' => S (max n' m')
  end.
Definition maxn m n := if m < n then n else m.
```

We shall use the following trivial lemmas.

```
Lemma ltn_add_le m1 m2 n1 n2: m1 < n1 -> m2 <= n2 -> m1 + m2 < n1 + n2.
Lemma ltn_add_el m1 m2 n1 n2: m1 <= n1 -> m2 < n2 -> m1 + m2 < n1 + n2.
Lemma ltn_add_ll m1 m2 n1 n2: m1 < n1 -> m2 < n2 -> m1 + m2 < n1 + n2.
```

We shall also use these ones. The first expression in the third lemma is $(n+1)(n'+1)-1$.

```
Lemma ltn_simpl1 n n': ((n' + n).+1 < n) =  false.
Lemma eqn_simpl1 n n': ((n' + n).+1 == n) = false.
Lemma ltn_simpl2 n n' n'':
  (n * n' + n + n' < n * n'' + n + n'') =  (n' < n'').
Lemma eqn_simpl2 n n' n'':
   (n * n' + n + n' == n * n'' + n + n'') =  (n' == n'').
```

1.3 Accessibility

In [1], Ackerman considers a set of ordinals, ordered by $\leq$, with smallest element 1, and defines accessibility as follows: Let $\mathfrak{A}(\alpha)$ denote "the property $\mathfrak{A}$ applied to α", $\mathfrak{K}_x(\alpha,\mathfrak{A}(x))$ denote "the property $\mathfrak{A}$ applied to all ordinals $< x$", and $\mathfrak{B}_x(\alpha,\mathfrak{A}(x))$ denote "if $\mathfrak{K}_x(\beta,\mathfrak{A}(x))$ holds, then $\mathfrak{A}(\beta)$ holds also, provided that $\beta \leq \alpha$". Let's assume moreover that $\mathfrak{A}(1)$ and $\mathfrak{B}_x(\alpha,\mathfrak{A}(x))$ imply $\mathfrak{K}_x(\alpha+1,\mathfrak{A}(x))$, via a constructive procedure that uses no further properties of $\mathfrak{A}$. Ackermann says that α is accessible via $\mathfrak{A}$, and notes that this is independent of $\mathfrak{A}$, thus says that α is accessible.

For simplicity, let's write $A(\alpha)$ instead $\mathfrak{A}(\alpha)$, $K(\alpha)$ instead of $\mathfrak{K}_x(\alpha,\mathfrak{A}(x))$, and $B(\alpha)$ instead of $\mathfrak{B}_x(\alpha,\mathfrak{A}(x))$. Then $K(\alpha)$ is $\forall x < \alpha, A(x)$, and $B(\alpha)$ is $\forall x \leq \alpha, K(x) \implies A(x)$. Assume $A(1)$. The accessibility of α is now: $B(\alpha) \implies K(\alpha+1)$. Given that $\alpha+1$ is the successor of α, the conclusion is $\forall x \leq \alpha, A(x)$.

Now Ackermann shows:

TI. 1 is accessible. Proof: $x \leq \alpha$ is equivalent to $x = 1$, and $A(1)$ is an assumption.

TII. If α is accessible, and $\beta < \alpha$, then β is accessible. Proof. Assume $B(\beta)$. Define $A'(x)$ to be true when $x > \beta$, and $A(x)$ otherwise. Then $B'(\alpha)$ holds. Since α is accessible, we get $K'(\alpha+1)$, i.e., $A'(x)$ whenever $x \leq \alpha$. If $x \leq \beta$, we deduce $A(x)$, thus $K(\beta+1)$.

TIII. If all ordinals less than α are accessible, so is α. Proof. Assume $B(\alpha)$ and $\gamma < \alpha$. Then $B(\gamma)$ holds. Since γ is accessible, we get $K(\gamma+1)$ thus $A(\gamma)$. Thus $K(\alpha)$ holds. Our assumption says $A(\alpha)$. Thus $\gamma \leq \alpha$ implies $A(\gamma)$. Qed.

Note that TIII implies TI. We may define accessibility via TII and TIII. This eliminates the need to define the successor $\alpha+1$. No property of $<$ is used here. Instead of saying that all ordinals are accessible, we say that $<$ is *well-founded*. In this case, we have a new principle of induction: Let $H(\alpha)$ be $\forall\alpha, K(\alpha) \implies A(\alpha)$. This can be restated as $\forall\alpha, B(\alpha)$. If every ordinal is accessible, it follows that every ordinal satisfies $A(\alpha)$. Thus $(\forall\alpha, H(\alpha)) \implies (\forall\alpha, A(\alpha))$. Example (from the standard library for integers, written as $\forall\alpha, A, (\forall\alpha, H(\alpha)) \implies A(\alpha)$):

```
Lemma lt_wf_ind :
  forall n (P:nat -> Prop), (forall n, (forall m, m < n -> P m) -> P n) -> P n.
```

Assume now that A is any type, and R a relation on A, denoted by $x \prec y$. We say that x is *accessible* for R, denoted $A_R(x)$ if any y such that $y \prec x$ is accessible. We say that R is *well-founded* if any x is accessible. We show here the definition and the induction principle.

```
Inductive Acc (x: A) : Prop :=
     Acc_intro : (forall y:A, R y x -> Acc y) -> Acc x.
Definition well_founded := forall a:A, Acc a.
```

```
Hypothesis Rwf : well_founded.
Theorem well_founded_induction_type :
  forall P:A -> Type,
    (forall x:A, (forall y:A, R y x -> P y) -> P x) -> forall a:A, P a.
```

Example. Let's show that the relation < on **N** is well-founded. We take an integer n and show by induction that it is accessible. The result is obvious for $n = 0$, as $m < 0$ is absurd. Assume now (H) that every $y < n$ is accessible. We want to show that every $y < n+1$ is accessible. The result holds by (H) if $y < n$. But otherwise $y = n$, and (H) just says that n is accessible.

```
Lemma lt_wf: well_founded (fun (a b:nat) => a < b).
Proof.
move => n; split;elim: n; first by move => y ; rewrite ltn0.
move => n H y; rewrite ltnS leq_eqVlt; case /orP; first by move => /eqP ->.
by apply: H.
Qed.
```

If R is well-founded, then there is no function $\mathbf{N} \to$ A such that $f(i+1) \prec f(i)$. Proof. Let P(a) be: if a is accessible, then a is not of the form $f(i)$. Assume a accessible and $a = f(i)$; let $b = f(i+1)$; we have $b \prec a$ so that b is accessible and of the form $f(j)$. This shows P by transfinite induction. Let $a = f(0)$. As R is well-founded, a is accessible, contradicting P(a).

```
Theorem not_decreasing :
  ~ (exists f : nat -> A, (forall i:nat, R (f i.+1) (f i))).
Proof.
case => f dec.
pose p a := Acc R a -> ~ exists i, a = f i.
have H: forall a, p a.
  move => a; apply: (well_founded_ind W p) => x Hx ax [i egi].
  move: (dec i); rewrite - egi => H1; move: (Hx _ H1 (acc_imp H1 ax)).
  by case; exists (i.+1).
move: (H _ (W (f 0))); case; by exists 0.
Qed.
```

1.3.1 Definition by transfinite induction, example

One can define functions by transfinite induction. This is not completely obvious, so let's start with an example (Exercise 15.14 of [2]). We want

(Ef1) $$f(0) = f(1) = 0, \qquad f(n+1) = 1 + f(1 + f(n)) \quad (n > 0).$$

Let P(n) be some property of $f(n)$ (for instance an algorithm for computing the value). One shows that P holds by transfinite induction, that is, we assume P(k) for all $k \le n$, and deduce P(n+1). This works provided that the arguments of f are $\le n$. An analysis of the specifications shows that P(k) must imply

(C(v,k)) $$k = 0 \text{ or } v < k \quad (v = f(k)).$$

It happens that $f(n) = n - 1$, and this can be shown directly by induction.

```
Definition f_spec f n :=
   if n is m.+2 then  (f (f m.+1).+1 ).+1 else 0.
Lemma f_spec_simp f n: (forall n, f n = f_spec f n) -> f n = n.-1.
```

Assume that $f(k) < k$ is obvious. Then an easy way to define f is to consider an auxiliary function $f_1(x,p)$, where p is the restriction of f to arguments $< x$, and use it to define $f(x)$, see example below. As $f(k) < k$ is non-obvious, the function p, as well as $f_1(x,p)$, returns a value and a proof that the value is not too big. The solution of [2] is to introduce a function, with the following type:

```
Definition f_aux :
  forall x, (forall z, z < x -> {y:nat | z = 0 \/ y < z})
      {y:nat | x = 0 \/ y < x}.
```

The object in braces that appears here is a sigma type. It is a record that holds a value v and a proof that $C(v,x)$ holds. One can destruct the structure via `case`, or access its field via `match`; the value can be obtained by `sval`.

The function $f_1(x,p)$ has the form: if x is zero or one, then return zero, and a proof that it satisfies C; if $x = n+1$, $(n > 0)$, use p in order to get a value y_1 (corresponding to $f(n)$), and a constraint $C(y_1)$; use $C(y_1)$ and p again in order to get a value y_2 (corresponding to the second call of f) and a constraint $C(y_2)$, compute a value y, and a proof of $C(y)$.

Instead of giving the value of f_1 as a lambda-term, we construct it using tactics. For instance, the solution of [2] uses `auto`, `refine` and `omega`. However, this definition might be so complicated that nothing can be proved about the function f.

We use here a variant, where $C(y,x)$ is written as $y \le x-1$. Note that C is never used when the second argument is zero; so that our constraint is $C(y,x+1)$, namely $y \le x$. We need to show that this implies $y+2 \le x+2$. This is trivial; however, proving it inside f_1 makes it too complicated. This is why we define an auxiliary function f_0.

```
Lemma f0 n p: p <= n -> p.+2 <= n.+2.
Proof. by rewrite ltnS ltnS. Qed.

Definition f1 :
  forall x, (forall z, z < x -> {y:nat |y <= z.-1}) ->
  {y:nat | y <= x.-1}.
Proof.
case; [by exists 0 | case; first by exists 0 ].
move => n Hr.
move: (Hr _ (ltnSn n.+1)) => [y1 h1].
move: (Hr _ (f0 h1)) =>  [y2 h2].
exists y2.+1; apply: (leq_trans h2 h1).
Defined.
```

One can define f_2 using `well_founded_induction` or a variant `Fix`. Both these functions take five arguments. The first argument is implicit, it is the type of the arguments of f; the second argument is implicit, it is a relation $<$, the third argument says that $<$ is well founded; then comes P (here the sigma-type) that can be deduced from the last argument F. Here $F(x,p)$ returns an object of type $P(x)$, provided that p says: for every y, if $y < x$ then $P(y)$ holds.

Assume that F satisfies the following property:

(Fext) $$(\forall\, y, H : y < x \implies p(y,H) = p'(y,H)) \implies F(x,p) = F(x,p').$$

This means that we get the same value if we replace p by a function that has the same behavior. In particular, if f_2 is defined via `Fix` and F, then $f_2(x) = \mathrm{F}(x, p')$, where p' says $f_2(y)$ for every $y < x$. The lemma is called `Fix_eq`. In the lemma `f_eqn` we do not specify the first argument of $\mathrm{F}(x, p')$ as it is implicit. Moreover, for the second argument p', there is no need to specify $y < x$, it is inferred by Coq.

The non-trivial point in the proof is to show (Fext). It is trivial for $x = 0$ and $x = 1$. The assumption $p = p'$ has to be used twice, as f appears twice in (Ef1). The first use is obvious; in order to use it a second it is necessary to introduce the variable y; and this is impossible if the body of f_1 is too complex. Here, a case analysis on the first sigma-type suffices.

```
Definition f2 := Fix lt_wf _ f1.

Lemma f_eqn x: f2 x =  f1 (n:=x) (fun y _ => f2 y).
Proof.
move: x; apply: (Fix_eq lt_wf) => n A B Hp.
rewrite /f1 /psum; congr S; apply: eq_big => // [] [i lin] _ /=.
by move: lin;rewrite /extension1; case ( ltnP i n) => //.
Qed.
```

We have now a function that returns a value and a proof. If we take just the value we get a function f that satisfies (Ef1). This is obvious if the argument is zero or one. Otherwise, we rewrite the previous equation in $f(n+2)$, destruct the first sigma-type in order to get a value $y_1 \le n$, then the second sigma-type in order to a get value $y_2 \le y_1$. Here $y_1 = f(n+1)$ and y_2 is $f(y_1)$. It suffices to unfold the two `sval` that have appeared.

```
Definition f (x:nat): nat := sval (f2 x).

Lemma f_correct n: f n = f_spec f n.
Proof.
case: n => //; case => // n.
rewrite /f_spec /f f_eqn /f1.
by case: (f2 n.+1) => y1 H1; case: (f2 y1.+1) => y2 H2.
Qed.
```

1.3.2 Definition by transfinite induction, variant

Consider the following specification

(Ef2) $$f(0) = f(2) = 0, \qquad f(2n+2) = 1 + f(2 + 2f(2n)) \quad (n > 0, \text{ even}).$$

The function $f(2n)$ satisfies equation (Ef1) studied above, so that we have $f(n) = (n-1)/2$ for even n. This can be restated as $f(2k+2) = k$. In the Coq code given below, we write the specification as $f(n) = \mathrm{F}(f, n)$ where F is a total function (defined for all n). Any solution of this equation satisfies also $f(2k+3) = k$.

```
Definition f_spec f n :=
   if n is m.+4 then  (f (f (m.+2)).*2.+2 ).+1 else 0.

Lemma f_spec_simp f n: ~~ odd n ->(forall n, ~~odd n -> f n = f_spec f n)
   -> f n = (n.-1)./2.
Lemma f_spec_simp1 f n: (forall n, ~~odd n -> f n = f_spec f n)
   -> f (n.*2.+2) = n.
Lemma f_spec_simp2 f n: (forall n,  f n = f_spec f n) ->  f(n.*2.+3) = n.
```

We define our function by transfinite induction using the well-founded relation $n <_e m$, saying that $n < m$, both arguments are even (we prove by induction on n, that n and $n+1$ are accessible, one of these relations being trivial). We show some trivial lemmas, in particular, show that being even is decidable.

```
Definition lte n m := [&& ~~ odd n, ~~ odd m & n < m].
Lemma lte_wf: well_founded lte.

Lemma f0a y n: odd n = false ->  odd n.+2 \/ y <= (n.+2)./2.-1 ->
   y <= n./2 /\ lte (y.*2).+2 n.+4.
Lemma f0b a b: odd a.*2.+2 \/ b <= (a.*2.+2)./2.-1 -> b <= a.
Lemma f0c n:  odd n = false -> lte n.+2 n.+4.
Lemma odd_dec n : {odd n} + {odd n = false}.
```

Consider now $q(x)$, the sigma-type whose value y satisfies x is odd or $y \leq (n-1)/2$; we define a function $f_1(x,p)$ whose type is $q(x)$, given that p says $q(z)$ for every z such that $z <_e x$. Our function returns zero when x is odd or < 4, otherwise, it is as above. Let f_2 be the function obtained by applying `Fix` to f_1 and $<_e$ and $f(x)$ the value of the record returned by $f_2(x)$. The proofs are as before, except that case $n = 0$, $n = 1$, $n = 2$ and $n = 3$ are trivial, and when n is odd

```
Definition f1 :
  forall x, (forall z, lte z x -> {y:nat | odd z \/ y <= (z./2).-1}) ->
  {y:nat | odd x \/ y <= (x./2).-1}.
Proof.
case; first by exists 0; right.
case; first by exists 0; left.
case; first by exists 0; right.
case; first by exists 0; left.
move => n  Hr.
case (odd_dec n) => on; first by exists 0; left; rewrite /= on.
move: (Hr _ (f0c on)) => [y1 h1].
move: (f0a on h1) => [sa sb].
move: (Hr _ sb) => [y2 h2].
exists y2.+1; right;  apply: (leq_trans (f0b h2) sa).
Defined.


Definition f2 := Fix lte_wf _ f1.
Definition f (x:nat): nat := sval (f2 x).

Lemma f_eqn x: f2 x =  f1  (fun y _ => f2 y).
Proof.
move: x; apply: (Fix_eq lte_wf).
case => //; case => //; case => //; case => //.
move => n p p' Hp; rewrite /f1; case: (odd_dec n) => // on.
rewrite Hp; case: (p' n.+2 (f0c on)) => y Hy /=.
by case: (f0a on Hy) => a b; rewrite Hp.
Qed.

Lemma f_correct n: ~~odd n ->  f n = f_spec f n.
Proof.
case: n => //; first by rewrite  /f /= f_eqn.
case => //; case; first by rewrite  /f /= f_eqn.
case => // n; rewrite /f_spec /f f_eqn /f1 /=.
```

```
case: (odd_dec n) => a b.
 by  move: (negbNE (negbNE b)); move /negP; case.
by case:(f2 n.+2) => x p /=;case: (f0a a p) => y q; case:(f2 x.*2.+2).
Qed.
```

1.3.3 Definition by transfinite induction, third example

We consider now a third example

$$f(n) = 1 + \sum_{k<n} f(k). \tag{Ef3}$$

There is obviously a unique solution, satisfying $f(n+1) = 2f(n)$, so $f(n) = 2^n$ by induction. Let's write the equation as $f(n) = \mathrm{F}(f,n)$.

We use here the two libraries `fintype` and `bigop`, they allow us to define $\mathrm{F}(f,n)$. Here k has type I_n (this is a record, with two fields, a value and a proof that the value is less than k). If f has type $\mathrm{I}_n \to \mathbf{N}$, then $\mathrm{F}(f,n)$ is defined (but not $f(n)$). We could proceed by induction as follows: for each n we have a function f_n, defined on I_n. For $n = 0$, the function is trivial as its domain is empty. We define $f_{n+1}(x)$ to be $f_n(x)$ when $x < n$ and $\mathrm{F}(f_n, n)$ when $x = n$.

We proceed here by transfinite induction. We first notice that if f has type $\mathbf{N} \to \mathbf{N}$, then $\mathrm{F}(f,n)$ is also defined, since k is automatically coerced into an integer. Moreover, $\mathrm{F}(f,n)$ depends only on the values $f(k)$ for $k < n$. We consider here $f_1(x,p)$, whose value is $\mathrm{F}(n,f')$, where p provides a value for every $k < n$, and f' is any total function that returns the same value. The conversion $p \mapsto f'$ has to be simple, for otherwise proving that f satisfies the fix-point equation becomes too complicated. For instance, we cannot directly use `ltnP`; instead we use it to construct a decision procedure that checks whether $k < n$ is true or false.

```
Definition psum (f: nat -> nat) n := \sum_(i< n) (f i).
Definition f_spec f:= forall n, f n = (psum f n).+1.

Lemma f_spec_simp f n: f_spec f -> f n = 2 ^ n.

Lemma psum_exten n f g :
   (forall k, k < n -> f k = g k) -> (psum f n).+1 = (psum g n).+1.
Lemma lt_dec n m: {n <m} + {~~ (n < m) }.
Proof. by case: (n<m);  [ left | right ]. Qed.


Definition extension  (n : nat) (p : forall k : nat, k < n -> nat) k :=
  match lt_dec k n with
    | left x =>  p k x
    | _ => 0 end.
```

We define f_2 as the fix-point of f_1, and $f(x)$ as $f_2(x)$. That f satisfies the equation is easy. Notes. The quantity $f_2(x)$ is the application of some constant function to x. Specifying the return type in f forces evaluation of this function call. The first argument of f_1 is implicit (it can be deduced from $z < n$); In the lemma `f_eqn`, the second argument of the second argument of f_1 is unused, and we replace it by a wildcard. However the type $y < n$ provides the value of the first argument; for this reason we give it in the form `(n:=x)`.

```
Definition f1 (n : nat) (h : forall z : nat, z < n -> nat) :=
    (psum (extension h) n).+1.
```

```
Definition f2 := Fix lt_wf _ f1.
Definition f (x:nat): nat := f2 x.

Lemma f_eqn x: f2 x =  f1 (n:=x) (fun y _ => f2 y).
Proof.
move: x; apply: (Fix_eq lt_wf) => n A B Hp; apply: psum_exten.
move => k _; rewrite /extension;case: (lt_dec k n) => // a; apply: Hp.
Qed.

Lemma f_correct: f_spec f.
Proof.
move => n; rewrite /f f_eqn /f1; apply: psum_exten => k kn.
by rewrite /extension; case: (lt_dec k n) => //; rewrite kn.
Qed.
```

We shall consider in what follows a relation $\prec$ that is not well founded; however we assume , that for some property P, any x satisfying P is accessible via $a \prec_{\mathrm{P}} b$, which is "P(a) and P(b) and $a \prec b$". We then get the following induction principle.

```
Section restricted_recursion.

Variables (A:Type)(P:A->Prop)(R:A->A->Prop).
Definition restrict a b := [/\ P a,  R a b & P b].
Definition well_founded_P := forall a, P a -> Acc restrict a.

Lemma P_well_founded_induction_type :
       well_founded_P  ->
       forall Q : A -> Type,
       (forall x : A, P x -> (forall y : A,P y-> R y x -> Q y) -> Q x) ->
       forall a : A, P a -> Q a.
End restricted_recursion.
```

1.4 Additional lemmas

We state here some trivial lemmas whose usage can simplify some proofs. The first says that "if p then true else false" can be simplified to p (when p is a boolean). The second lemma says that $(a+b)+1$ is neither equal to a, nor less than a (by commutativity, a can be replaced by b). We state two lemmas that say, essentially, that $f(x) = (a+1)(x+1)-1$ is strictlty increasing.

```
Lemma if_simpl (p: bool): (if p then true else false) = p.
Lemma ltn_simpl1 n n': ((n' + n).+1 < n)%N =  false.
Lemma eqn_simpl1 n n': ((n' + n).+1 == n)%N = false.
Lemma ltn_simpl2 n n' n'':
  (n * n' + n + n' < n * n'' + n + n'') =  (n' < n'').
Lemma eqn_simpl2 n n' n'':
   (n * n' + n + n' == n * n'' + n + n'') =  (n' == n'').
Lemma expn_ge2 n m: ((n.+2 ^ m.+1)  = (n.+2 ^ m.+1).-2.+2).
```

Chapter 2

The first model

2.1 Equality

Our first type will be denoted `T1`. It has two constructors, `zero`, denoted by 0, and '`cons a n b`', denoted by $[a,n,b]$. We can associate an ordinal $O(x)$ to each x of type T_1: if x is 0, we associate then $O(0) = 0_o$, and $O([a,n,b]) = \omega_o^{O(a)} \cdot (n+1) + O(b)$. (here 0_o is the ordinal zero, and ω_o is the least infinite ordinal).

```
Inductive T1 : Set :=
  zero : T1
| cons : T1 -> nat -> T1 -> T1.
```

Our first operation consists in defining a boolean equality $x \equiv y$, and show that this relation is equivalent to $x = y$. We use then the mechanism of canonical structures provided by SSREFLECT. This allows us to use the notation `x==y` for ordinals. The equivalence between `x==y` and `x=y` is provided by `eqP`, whatever the type of x and y. We shall sometimes use the fact that $[a,n,b] \equiv [a',n',b']$ is equal to "$a \equiv a'$, $n \equiv n'$ and $b \equiv b'$". (in the current context, only natural integers, booleans and ordinal have a boolean equality). In particular $[a,n,b] \equiv [a',n',b']$ is equivalent to $a = a'$, $b = b'$ and $n = n'$.

We shall define below a comparison `T1le`, based on the integer comparison `leq`. It would be nice if we could use the same notation $x \leq y$. This is not possible. For this reason, we use the scoping mechanism of COQ. We use `X%ca` and `X%N` to force the interpretation of X to be in the scope of our ordinals, or the scope of natural integers. Moreover, the default scope will be that the ordinals.

```
Fixpoint T1eq x y {struct x} :=
  match x, y with
  | zero, zero  => true
  | cons a n b, cons a' n' b' => [&& T1eq a a', n== n' & T1eq b b' ]
  | _, _ => false
end.

Lemma T1eqP : Equality.axiom T1eq.
Canonical T1_eqMixin := EqMixin T1eqP.
Canonical T1_eqType := Eval hnf in EqType T1 T1_eqMixin.

Implicit Arguments T1eqP [x y].
```

```
Prenex Implicits T1eqP.

Delimit Scope cantor_scope with ca.
Open Scope cantor_scope.

Lemma T1eqE a n b a' n' b':
  (cons a n b == cons a' n' b') = [&& a  == a', n== n' &  b == b' ].
```

Let's start with some definitions. We define an injection $\mathbf{N} \to T_1$ by $F(0) = 0$ and $F(n+1) = [0,n,0]$. We shall write $\phi_0(x)$ instead of $[x,0,0]$. Note that $F(1) = \phi_0(0)$. This will be denoted by one. The quantity $\phi_0(\phi_0(0))$ will be denoted by ω. We the previous notations, we have $O(\omega) = \omega_0$, and $O(F(n))$ is a finite ordinal. The logarithm of $[a,n,b]$ will be a. For completeness, the logarithm of zero will be zero. We say that x is additive principal, in short AP, if it has the form $[a,0,0]$.

```
Definition phi0 a := cons a 0 zero.
Definition one := cons zero 0 zero.
Definition T1omega := phi0 (phi0 zero).
Definition T1bad := cons zero 0 T1omega.
Definition T1nat (n:nat) : T1 :=
  if n is p.+1 then cons zero p zero else zero.
Definition T1log a := if a is cons a _ _ then a else zero.
Definition T1ap x := if x is cons a n b then ((n==0) && (b==zero)) else false.

Notation "\F n" := (T1nat n)(at level 29) : cantor_scope.
```

Some trivial results.

```
Lemma T1F_inj: injective T1nat.
Lemma T1phi0_zero : phi0 zero = \F 1.
Lemma T1phi0_zero' : phi0 zero = one.
Lemma T1log_phi0 x : T1log (phi0 x) = x.
Lemma T1ap_phi0 x: T1ap (phi0 x).
```

2.2 Ordering

Let's define an ordering on our ordinals. We assume that zero is the least ordinal. We compare $[a,n,b]$ and $[a',n',b']$ lexicographically.

```
Fixpoint T1lt x y {struct x} :=
  if x is cons a n b then
    if y is cons a' n' b' then
      if a < a' then true
      else if  a == a' then
         if (n < n')%N then true
         else if (n == n')  then b < b' else false
         else false
      else false
  else if y is cons a' n' b' then true else false
where  "x < y" := (T1lt x y) : cantor_scope.

Definition T1le (x y :T1) := (x == y) || (x < y).
```

```
Notation "x <= y" := (T1le x y) : cantor_scope.
Notation "x >= y" := (y <= x) (only parsing) : cantor_scope.
Notation "x > y"  := (y < x) (only parsing)  : cantor_scope.
```

We give here a variant of $[a,n,b] \leq [a',n',b']$.

```
Lemma T1le_consE  a n b a' n' b':
 (cons a n b <= cons a' n' b') =
    if a < a' then true
      else if  a == a' then
         if (n < n')%N then true
         else if (n == n')  then b <= b' else false
         else false.
```

We state some trivial facts.

```
Lemma T1lenn x: x <= x.
Lemma T1ltnn x: (x < x) = false.
Lemma T1lt_ne a b : a < b -> (a == b) = false.
Lemma T1lt_ne' a b : a < b -> (b == a) = false.
Lemma T1ltW a b : (a < b) -> (a <= b).
Lemma T1le_eqVlt a b : (a <= b) = (a == b) || (a < b).
Lemma T1lt_neAle a b : (a < b) = (a != b) && (a <= b).

Lemma T1ltn0 x: (x < zero) = false.
Lemma T1le0n x: zero <= x.
Lemma T1len0 x: (x <= zero) = (x == zero).
Lemma T1lt0n x: (zero < x) = (x != zero).
Lemma T1ge1 x: (one <= x) = (x != zero).
Lemma T1lt1 x: (x < one) = (x==zero).
Lemma T1nat_inc n p : (n < p)%N = (\F n < \F p).
```

Our first non-trivial result is that, for ant a and b, we have $a < b$, $b < a$ or $a = b$; these cases being mutually exclusive. two district elements can uniquely be compared via $<$. We deduce some easy consequence, including `T1ltgtP` and friends, that depends on definitions not given here, but similar to those shown in the first chapter.

```
Lemma T1lt_trichotomy a b: [|| (a< b), (a==b) | (b < a)].
Lemma T1lt_anti b a: a < b -> (b < a) = false.
Lemma T1le_total m n : (m <= n) || (n <= m).
Lemma T1leNgt a b:  (a <= b) = ~~ (b < a).
Lemma T1ltNge a b:  (a < b) = ~~ (b <= a).
Lemma T1eq_le m n : (m == n) = ((m <= n) && (n <= m)).
Lemma T1leP x y : T1leq_xor_gtn x y (x <= y) (y < x).
Lemma T1ltP m n : T1ltn_xor_geq m n (n <= m) (m < n).
Lemma T1ltgtP m n : compare_T1 m n (m < n) (n < m) (m == n).
```

Our second non-trivial result is transitivity of $<$ (proof by induction on c).

```
Lemma T1lt_trans b a c: a < b -> b < c -> a < c.
Lemma T1lt_le_trans b a c: a < b -> b <= c -> a < c.
Lemma T1le_lt_trans b a c: a <= b -> b < c -> a < c.
Lemma T1le_trans b a c: a <= b -> b <= c -> a <= c.
```

We list some other trivial lemmas. Note that $a < [a,n,b]$ (proof by induction). When n and b are zero this reduces $a < \phi_0(a)$, to $a < \omega^a$ (ordinal power will be defined later on). Relaccl that ϵ_0 is tthe least ordinal x such that $x = \omega^x$. We deduce that it is not of the form $O(a)$. In fact, it is the least such ordinal.

```
Lemma head_lt_cons a n b: a < cons a n b.
Lemma phi0_lt a b: (phi0 a < phi0 b) = (a < b).
Lemma phi0_le a b: (phi0 a <= phi0 b) = (a <= b).
Lemma T1lt_cons_le a n b a' n' b': (cons a n b < cons a' n' b') -> (a <= a').
Lemma T1le_cons_le a n b a' n' b': (cons a n b <= cons a' n' b') -> (a <= a').
Lemma phi0_lt1 a n b a':(cons a n b < phi0 a') = (a < a').
```

2.3 Normal form

The comparison < of ordinals is not well-founded (consider the function f such that $f(0) = \omega$ and $f(i+1) = [0,0,f(i)]$).

```
Theorem lt_not_wf : ~ (well_founded T1lt).
```

We say that an ordinal is in *normal form*, in short NF, if it is zero, or $[a,n,b]$, where a and b are normal, and moreover $b < \phi_0(a)$. If $a = 0$, this says $b = 0$. If $b = [a',n',b']$, this implies $a' < a$. The quantity $\bar{\omega} = [0,0,\omega]$ introduced above is not NF and will be used as a counter example in a lot of cases. Note that $O(\bar{\omega}) = \omega$ so that O is not injective. However, it will becom injective when restriced to NF ordinals.

```
Fixpoint T1nf x :=
  if x is cons a _ b then [&& T1nf a, T1nf b & b < phi0 a ]
  else true.

Lemma T1nf_cons_cons a n a' n' b' : T1nf (cons a n (cons a' n' b')) -> a' < a.
Lemma T1nf_cons0 a n: T1nf a -> T1nf (cons a n zero).
Lemma T1nf_finite1 n b: T1nf (cons zero n b) = (b == zero).
Lemma T1nf_finite n b: T1nf (cons zero n b) -> b = zero.
Lemma T1nf_consa a n b: T1nf (cons a n b) -> T1nf a.
Lemma T1nf_consb a n b: T1nf (cons a n b) -> T1nf b.

Lemma T1nfCE: ~~(T1nf T1bad).
```

We show here that the relation "N(x) and N(y) and $x < y$", denoted $x <_N y$, is well-founded. We first give the proof, then some explanations.

```
Lemma nf_Wf: well_founded (restrict T1nf T1lt).
Proof.
have az: Acc (restrict T1nf T1lt) zero by split => y [_]; rewrite T1ltn0.
elim;[ exact az | move => a Ha n b _].
elim:{a} Ha n b => a Ha Hb n b.
case nx: (T1nf (cons a n b)); last by split => y [_ _]; rewrite nx.
move/and3P: (nx);rewrite -/T1nf; move => [na nb lba].
have aca: Acc (restrict T1nf T1lt) a by split.
have Hd: forall b, T1nf b -> b < phi0 a -> Acc (restrict T1nf T1lt)  b.
  case; [by move => _ _ ; apply: az | move => a' n' b' nx'].
  rewrite phi0_lt1 => aa'.
```

```
  by apply: Hb; rewrite /restrict  (T1nf_consa nx') na aa'.
have Hc: forall c, Acc (restrict T1nf T1lt) c ->
  T1nf (cons a 0 c)-> Acc (restrict T1nf T1lt) (cons a 0 c).
  move => c; elim => {c} c qa qb qc; split; case; first by move => _; apply: az.
  move => a'' n'' b'' [] sa /= ua /and3P [_ nc _];move/and3P:(sa) => [ra rb _].
  move: ua;case: (T1ltgtP a'' a) => ua ub.
  - by apply: Hb.
  - by case ub.
  - by move: ub sa; case ee:(n''==0); [rewrite ua (eqP ee) => ub; apply: qb | ].
elim: n b {nb lba} (Hd _ nb lba) nx => [ // | n He b]; elim.
move => c _ qb np; split; case; first by move => _; apply: az.
move => a'' n'' b'' [sa /= sb _];move /and3P: (sa) =>  [ra rb rc].
move: sb; case: (T1ltgtP a'' a) => sc sb;[ by apply: Hb | by case sb |].
move: sb; case: (ltngtP n'' n.+1); [rewrite ltnS leq_eqVlt | done | move ->].
  rewrite sc in rc; move => sb _; move: sa; case /orP: sb.
    move => /eqP ->; rewrite sc; apply: (He b'' (Hd _ rb rc)).
  move => qd qe.
  have nc0: T1nf (cons a n zero) by rewrite /= andbT.
  apply: (acc_rec (And3 qe _ nc0) (He _ az nc0)).
  by rewrite /= qd sc eqxx T1ltnn.
move => sb; move/and3P: np => [pa pb pc].
rewrite sc;apply: (qb _ (And3 rb sb pb)); rewrite -sc //.
Qed.

Lemma nf_Wf' : well_founded_P T1nf T1lt.
```

Let $A(x)$ denote the fact that x is accessible via $<_N$. We obviously have $A(0)$. We show $A(x)$ by induction on x; since tha case $x = 0$ is trivial, we show $[a,n,b]$, assume $A(a)$ and $A(b)$. The second assumption is useless, and we proceed by induction on $A(a)$. So we are to prove $A([a,n,b])$, assuming H_a that says that every y such that $y <_R a$ satisfies A (this is equivalent to $A(a)$), and H_b that says, whatever y, n, b, if $y <_N a$ and $N([y,n,b])$, then $A([y,n,b])$. We may assume $N([a,n,b])$, for otherwise the result is trivial. In particular, we have $N(a)$, $N(b)$ and $b < \phi_0(a)$. We first show H_d: if $b < \phi_0(a)$ and $N(b)$ then $A(b)$. We may assume b non-zero, say $b = [a',n',b']$, the first assumption is $a' < a$, and the result follows from H_b. We then show H_c: for every c, $A(c)$ and $N([a,0,c])$ imply $A([a,0,c])$. The proof is by induction on $A(c)$. So, instead of $A(c)$ we have q_a that says $y <_N c$ implies $A(y)$ and q_b that says: if $y <_N c$ and $N([a0,y])$ then $N([a0,y])$. We must show that all ordinals x such that $x < [a,0,c]$ are accessible; this holds trivially for zero, so consider $[a'',n'',b'']$. The relation $x < [a,0,c]$ is equivalent to either $a'' < a$ (case where we can apply H_b) or $a'' = a$, $n'' = 0$ and $b'' < c$ (case where we can apply q_b). Assumption H_d says $N(b)$, and we prove our result by induction on n (for any b such that $N(b)$). The case $n = 0$ is H_c; our induction assumption H_e says for every b, $A(b)$ and $N([a,n,b])$, implies $A([a,n,b])$. We must show: $A(b)$ and $N([a,n+1,b])$ implies $A([a,n+1,b])$, proof by induction on $A(b)$; thus we show that $x <_N [a,n+1,c]$ implies $A(x)$; the case $x = 0$ being trivial we may assume $x = [a'',n'',b'']$. We have q_b: whatever b', $b' <_N c$ and $A([a,n+1,b'])$ implies $A([a,n+1,b'])$. The relation $x <_N [a,n+1,c]$ holds when $a' < a$ (case where H_b applies), or when $a' = a$, $n' = n+1$, $b' < c$ (case where q_b applies), or when $a' = a$, $n' < n+1$. Assume first $n' = n$. We have $A(b')$ by H_d and we conclude by H_e. Otherwise $n' < n$ thus $x <_N [a,n,0]$; the result follows from $A([a,n,0])$, a consequence of H_e.

One deduces the following transfinite induction. Let $N'(x)$ denote "x is NF and $Q(x)$". Let $H(x)$ be the assumption that a sufficient condition for $P(x)$ to hold (given $N'(x)$) is: every y such that $N'(y)$ and $y < x$ satisfies P. Then $N'(x)$ and $H(x)$ implies $P(x)$. We have a variant where Q is trivial. We also state a structural induction principle. (These properties were shown by Castéran, they are not used in the following code).

```
Lemma T1transfinite_induction P:
  (forall x, T1nf x -> (forall y, T1nf y ->  y < x -> P y) -> P x) ->
  forall a, T1nf a -> P a.
Lemma T1transfinite_induction_Q (P: T1 -> Type) (Q: T1 -> Prop):
  (forall x:T1, Q x -> T1nf x ->
               (forall y:T1, Q y -> T1nf y ->  y < x -> P y) -> P x) ->
  forall a, T1nf a -> Q a -> P a.
Lemma T1nf_rect (P : T1 -> Type):
   P zero ->
   (forall n: nat,  P (cons zero n zero)) ->
   (forall a n b n' b', T1nf (cons a n b) ->
                        P (cons a n b) ->
                        b' < phi0 (cons a n b) ->
                        T1nf b' ->
                        P b' ->
                        P (cons (cons a n b) n' b')) ->
   forall a, T1nf a -> P a.
```

2.4 Successor

Let x be an ordinal; if non-zero, it has the form $[a,n,b]$. If b is non-zero, it has this same form. Thus x has the form $[a_0,n_0,[a_1,n_2,\dots[a_k,n_k,0]]]$. We say that a_0 is the first exponent, and a_k is the last one. If x is NF, then the sequence a_i is strictly decreasing. In particular, only the last exponent can be zero. The last coefficient $c(x)$ of x is n_k+1 when $a_k=0$, it is zero otherwise. We define the limit part $l(x)$ of x as x, when a_k is non zero, and otherwise the expression obtained from x by replacing $[a_k,n_k,0]$ by zero. We shall show below that if x is NF, then $x=l(x)+c(x)$.

We say that x is a *limit* ordinal if the last exponent is non-zero (in particular, x is non-zero). The quantity $c(x)$ is zero if and only if x is zero or limit. We say that x is a *successor* if the last exponent is zero and x is non-zero. This is the same as $c(x)\neq 0$. An ordinal is either zero, limit or a successor.

```
Fixpoint T1split x:=
 if x is cons a n b then
      if a==zero then (zero, n.+1) else
     let: (x, y) := T1split b in (cons a n x,y)
   else (zero,0).

Fixpoint T1limit x :=
  if x is cons a n b then
    if a==zero then false else (b== zero) || T1limit b
  else false.

Fixpoint T1is_succ x :=
  if x is cons a n b then  (a==zero) || T1is_succ b  else false.

Lemma split_is_succ x: ((T1split x).2 != 0) = (T1is_succ x).
Lemma split_limit x: ((T1split x).2 == 0) = ( (x==zero) || T1limit x).
Lemma split_limit1 x (y:= (T1split x).1): (y == zero) || (T1limit y).
Lemma limit_pr1 x: (x == zero) (+) (T1limit x (+) T1is_succ x).
```

We say that x is *finite* if if the first exponent is zero. This is the same as $l(x)=0$. If x^+ is the *successor* of x, then $l(x^+)=l(x)$ and $c(x^+)=c(x)+1$. The inverse of this operation is called

the *predecessor*. We have $l(x^-) = l(x)$ and $c(x^-) = c(x) - 1$. (These properties do not uniquely define the successor and the predecessor).

```
Definition T1finite x := if x is cons a n b then a == zero else true.

Fixpoint T1succ (c:T1) : T1 :=
  if c is cons a n b
     then if a == zero then cons zero n.+1 zero else cons a n (T1succ b)
  else one.

Fixpoint T1pred (c:T1) : T1 :=
  if c is cons a n b then
     if (a==zero) then \F n else (cons a n (T1pred b))
  else zero.

Lemma split_finite x: ((T1split x).1 == zero) = T1finite x.
Lemma split_succ x: let:(y,n):= T1split x in T1split (T1succ x) = (y,n.+1).
Lemma split_pred x: let:(y,n):= T1split x in T1split (T1pred x) = (y,n.-1).
Lemma split_le x :  (T1split x).1 <= x.
Lemma nf_split x : T1nf x -> T1nf (T1split x).1.
```

If F is the natural injection $\mathbf{N} \to \mathrm{T}_1$, then $\mathrm{F}(n)$ is finite for every n. Conversely, any finite normal ordinal has the form $\mathrm{F}(n)$. The successor of a finite ordinal is finite.

```
Lemma T1finite1 n: T1finite (\F n).
Lemma T1finite2 x: T1finite x -> T1nf x -> x = \F ((T1split x).2).
Lemma T1finite_succ x: T1finite x -> T1finite (T1succ x).
Lemma T1succ_nat n: T1succ (\F n) = \F (n.+1).
Lemma T1finite2CE: T1finite T1bad /\ forall n, T1bad <> \F n.
```

We have $x < x^+$ and $x^- \leq x$. If x is a successor, then $x^- < x$. If y is limit and $x < y$ then $x^+ < y$.

We show here that some quantities are NF. Assume that x has a predecessor y. In order to show that y is NF whenever x is NF, we first show $y < x$.

```
Lemma pred_le a: T1pred a <= a.
Lemma pred_lt a: T1is_succ a -> T1pred a < a.
Lemma succ_lt a: a < T1succ a.
Lemma limit_pr x y: T1limit x -> y < x -> T1succ y < x.

Lemma nf_omega : T1nf T1omega.
Lemma nf_phi0 a: T1nf (phi0 a) = T1nf a.
Lemma nf_finite n: T1nf (\F n).
Lemma nf_log a: T1nf a -> T1nf (T1log a).
Lemma nf_succ a: T1nf a -> T1nf (T1succ a).
Lemma nf_pred a: T1nf a -> T1nf (T1pred a).
```

One has $(x^+)^- = x$ and $(x^-)^+ = x$, provided that both arguments are NF, and x is a successor in the second case.

```
Lemma succ_p1 x: T1is_succ (T1succ x).
Lemma succ_pred x: T1nf x -> T1is_succ x -> x = T1succ (T1pred x).
Lemma pred_succ x: T1nf x -> T1pred (T1succ x) = x.

Lemma succ_predCE: T1is_succ T1bad /\ forall y, T1bad <> T1succ y.
Lemma pred_succ_CE: T1pred (T1succ T1bad) <>  T1bad.
```

If $x^+ < y^+$ then $x < y$. This is equivalent to : if $x \leq y$ then $x^+ \leq y^+$. Assume x and y normal. Then $x < y$ is equivalent to $x^+ < y^+$, and $x \leq y$ is equivalent to $x^+ \leq y^+$. Moreover $x \leq y$ is equivalent to $x < y^+$ and $x < y$ is equivalent to $x^+ \leq y$. As shown here, not all arguments need to be normal.

```
Lemma succ_inj x y: T1nf x -> T1nf y -> (T1succ x == T1succ y) = (x==y).
Lemma lt_succ_succ x y:  T1succ x < T1succ y -> x < y.
Lemma le_succ_succ x y: x <= y -> T1succ x <= T1succ y.
Lemma lt_succ_succE x y:
  T1nf x -> T1nf y ->  (T1succ x < T1succ y) = (x < y).
Lemma le_succ_succE x y:
  T1nf x -> T1nf y -> (T1succ x <= T1succ y) = (x <= y).
Lemma lt_succ_le_1 a b : T1succ a <= b ->  a < b.
Lemma lt_succ_le_2 a b:  T1nf a -> a < T1succ b ->  a <= b.
Lemma lt_succ_le_3 a b:  T1nf a -> (a < T1succ b) = (a <= b).
Lemma lt_succ_le_4 a b:  T1nf b -> (a < b) = (T1succ a <= b).

Lemma succ_injCE: one <> T1bad /\ (T1succ one = T1succ T1bad).
Lemma lt_succ_succCE: (one  < T1bad) &&  ~~ (T1succ one < T1succ T1bad).
Lemma lt_succ2CE: one < T1bad /\ T1bad < T1succ one.
```

Any ordinal is zero, limit or a successor. If x is limit and $y < x$ then $y^+ < x$.

```
Lemma phi0_log a: a < phi0 (T1succ (T1log a)).
Lemma limit_pr x y: T1limit x -> y < x -> T1succ y < x.
```

2.4.1 Addition

We define here addition and subtraction. Subtraction is a bit complex; in the case $a = 0$, if x is NF, then $b = 0$, thus $b - b' = 0$.

```
Fixpoint T1add (c1 c2 : T1) {struct c1}:T1 :=
  match c1,c2 with
 |  zero, y  => y
 |  x, zero  => x
 |  cons a n b, cons a' n' b' =>
       if a < a' then  cons a' n' b'
       else if a' < a then (cons a n (b + (cons a' n' b')))
       else  (cons a (n+n').+1 b')
 end
 where "a + b" := (T1add a b) : cantor_scope.

Fixpoint T1sub x y :=
 if x is cons a n b then
    if y is cons a' n' b' then
      if  x < y then zero
      else if a' < a then cons a n b
      else if (n < n')%N then zero
      else if (a==zero) then
         if (n <= n')%N then zero else cons zero ((n-n').-1) zero
      else if (n == n') then b - b' else cons a (n-n').-1 b
    else x
  else  zero
where  "x - y" := (T1sub x y):cantor_scope.
```

As stated above, $x+1$ is the successor of x. Note that $x-1$ is not the predecessor of x, but we have $x = (1+x)-1 = 1+(x-1)$.

We have two interesting properties: if $x = [a,n,b]$ then x is the sum of $[a,n,0]$ and b. More generally, if we split x as $l(x)$ and $c(x)$, we have $x = l(x)+c(x)$. Here $c(x)$ is an integer, and $l(x)$ a limit ordinal (or maybe zero).

```
Lemma succ_is_add_one a: T1succ a = a + one.
Lemma add1Nfin a:  ~~ T1finite a  -> one + a = a.
Lemma sub1Nfin a:  ~~ T1finite a  -> a -one  = a.
Lemma sub1a x: x != zero -> T1nf x -> x = one + (x - one).
Lemma sub1b x: T1nf x -> x = (one + x) - one.

Lemma T1add0n : left_id1zero T1add.
Lemma T1addn0: right_id zero T1add.
Lemma T1subn0 x: x - zero = x.
Lemma T1subnn x: x - x = zero.
Lemma add_int n m : \F n + \F m = \F (n +m)%N.
Lemma sub_int n m : \F n - \F m = \F (n -m)%N.
Lemma add_fin_omega n: \F n + T1omega = T1omega.
Lemma split_add x: let: (y,n) :=T1split x in  T1nf x ->
   (x == y + \F n) && ((y==zero) || T1limit y ).
Lemma add_to_cons a n b:
  b < phi0 a -> cons a n zero + b = cons  a n b.

Lemma addC_CE (a := one) (b := T1omega):
  [/\ T1nf a, T1nf b & a + b  <> b + a].
Lemma sub_1aCE (a:=  cons zero 0 T1bad) : one + (a - one) != a.
Lemma sub_1bCE (a:=  cons zero 0 T1bad) : (one + a - one) != a.
```

If $x = \phi_0(a)$, then $b < x$ and $c < x$ imply $b+c < x$. The converse holds, when x is non-zero. We call these numbers "additive principal". Proof. Assume $x = [a,n,b]$. If $n = m+1$, we consider $y = [a,m,b]$; otherwise $y = [a,0,0]$. In both cases, $y < x$ and $y+y < x$ is false. If x is NF, it suffices to consider b and c NF (same argument).

If a and b are NF, so is $a+b$. The sum of $[a,n,0]$ and b is $[a,n,b]$, provided that $b < \phi_0(a)$.

```
Lemma ap_pr0 a (x := phi0 a) b c:
     b < x -> c < x -> b + c < x.
Lemma ap_pr1 c:
   (forall a b,  a < c -> b < c -> a + b < c)  ->
   (c== zero) || T1ap c.
Lemma ap_pr2 c:
   T1nf c -> c <> zero ->
   (forall a b, T1nf a -> T1nf b ->  a < c -> b < c -> a + b < c)  ->
   T1ap c.
Lemma ap_pr3 a b (x := phi0 a): b < x -> b + x = x.
Lemma ap_pr4 x: (forall b, b < x -> b + x = x) -> (x == zero) || T1ap x.
Lemma nf_add a b: T1nf a -> T1nf b -> T1nf (a + b).
Lemma add_to_cons a n b:
   b < phi0 a -> cons a n zero + b = cons  a n b.

Lemma ap_pr2CE (c := cons T1bad 1 zero):
   (forall a b, T1nf a -> T1nf b ->  a < c -> b < c -> a + b < c).
```

If a and b are two ordinals, then $a \le a+b$ and $b \le a+b$. If b is non-zero then $a < a+b$. Note: it is possible that $a+b = b$. In fact, every b such that $b \ge a\cdot\omega$ is a solution (multiplication

is defined later on).

```
Lemma T1add_eq0 m n:  (m + n == zero) = (m == zero) && (n == zero).
Lemma add_le1 a b: a <= a + b.
Lemma add_le2 a b: b <= a + b.
Lemma add_le3 a b: a = a + b -> b = zero.
Lemma add_le4 a b: b != zero -> a < a + b.
Lemma T1ge1 a: a != zero = (one <= a).
Lemma phi0_inj a b: (phi0 a == phi0 b) = (a==b).
Lemma phi0_le a b: (a <= b) = (phi0 a <= phi0 b).
Lemma add_simpl1 a n b n' b': a != zero ->
   cons a n b + cons zero n' b' =  cons  a n (b + cons zero n' b').
Lemma add_simpl2  n b a' n' b': a' != zero ->
   cons zero n b + cons a' n' b' = cons a' n' b'.
```

We have $(a+b)-a=b$, when b is NF. In particular $a+b=a+b'$ implies $b=b'$. We have $b=a+(b-a)$ whenever $a\leq b$, b is normal. Another nontrivial result is associativity of addition.

```
Lemma minus_lt a b: a < b -> a - b = zero.
Lemma minus_le a b: a <= b -> a - b = zero.
Lemma T1sub0 a: a - zero = a.
Lemma nf_sub a b: T1nf a -> T1nf b -> T1nf (a - b).
Lemma sub_le1 a b : T1nf a -> (a - b) <= a.
Lemma sub_pr1 a b: T1nf b -> a <= b -> b = a + (b - a).
Lemma sub_pr a b: T1nf b -> (a + b) - a = b.
Lemma sub_pr1r a b: T1nf a -> a - b = zero -> a <= b.
Lemma sub_nz a b: T1nf b -> a < b -> (b - a) != zero.
Lemma sub_pr1' b: b != zero -> T1nf b ->  (b = one + (b - one)).

Lemma add_inj a b c : T1nf b -> T1nf c -> a + b = a + c -> b = c.
Lemma plus_assoc c1 c2 c3: c1 + (c2 + c3) = (c1 + c2) + c3.

Lemma T1le_add2l  p m n : (p + m <= p + n) = (m <= n).
Lemma T1lt_add2r  p m n : (m + p  < n + p ) ->  (m < n).
Lemma T1lt_add2l  p m n : (p + m < p + n) = (m < n).
Lemma T1le_add2r  p m n : (m <=n) -> (m + p  <= n + p).
Lemma T1eq_add2l  p m n : (p + m == p + n) = (m == n).
```

We give here examples of some properties.

```
Lemma T1finite2CE (x := cons zero 0 T1omega):
   T1finite x /\ forall n, x <> \F n.
Lemma plus_not_comm (a := one) (b := T1omega):
  [/\ T1nf a, T1nf b & a + b  <> b + a].
Lemma omega_minus_one : T1omega - one = T1omega.
Lemma sub_nzCE (a := one) (b := (cons zero 0 one)):
   (a < b) && (b-a == zero).
Lemma sub_pr1CE: (one <= T1bad) &&  (T1bad != one + (T1bad - one)).
Lemma sub_pr1rCE (a := T1bad) (b := one) : (a - b == zero) && (b < a).
Lemma sub_pr1''CE (a:=  cons zero 0 T1bad) : (one + a - one) != a.
```

One can define the *normal form* $\mathrm{N}(x)$ of x by $\mathrm{N}([a,n,b])=[\mathrm{N}(a),n,0]+\mathrm{N}(b)$. Then $\mathrm{N}(x)$ is NF, and equal to x, when x is NF. This function is not really interesting (for instance, it is not compatible with the successor function).

```
Fixpoint toNF x :=
  if x is cons a n b then (cons (toNF a) n zero) + toNF b else zero.

Lemma nf_toNF x: T1nf (toNF x).
Lemma toNF_nf x: T1nf x -> toNF x = x.
Lemma toNF_nz x : toNF x = zero -> x = zero.
Lemma toNF_mon x : x <= toNF x.

Lemma toNF_succ (x := cons zero 0 one): toNF (T1succ x) != T1succ (toNF x).
Lemma toNF_ex1 x: toNF (cons zero 0 x) = one + toNF x.
Lemma toNF_ex2: toNF (cons one 0 T1omega) = cons one 1 zero.
```

2.5 Limit ordinals

We shall consider here the problem of the supremum of a set of ordinals E. Assume first that we have solved our major problem, that of finding an injection of T_1 into the ordinals. Let f_1 be the injection, O_1 the image. Given a set of ordinals E, one can well-order it, and consider the ordinal sum x of the elements of E for this ordering. This gives an upper bound x of E. The set of ordinals z that are $\leq x$ and upper bounds of E is non-empty, thus has a least element, called the supremum of E and denoted sup(E); it is clearly independent of x. Let's denote by α the supremum of O_1. Assume that E is a subset of T_1; by f_1 we get a set E_1, and a supremum x_1. We have obviously $x_1 \leq \alpha$. We may have equality; in this case E is unbounded, and has no supremum in T_1; we shall give an example below. The fact that $<$ is well-founded on T_1 asserts that there exists f_1 such that O_1 is the set of all ordinals $< \alpha$; in this case $x_1 < \alpha$ says $x_1 \in O_1$, so that there is $x \in T_1$ such that $x_1 = f(x)$. In this case, x is the supremum of E.

Let's assume the Excluded Middle Principle, under the form: for any proposition P on T_1, either there is an NF x satisfying P, or there is none. Let Q by a proposition satisfied by y and $P(x)$ be $x < y$ and $Q(x)$; by transfinite induction there is a least z satisfying Q (in the first case, we apply the assumption to x, in the second case y is the least element).

```
Fixpoint omega_tower (n:nat) : T1 :=
  if n is p.+1 then phi0 (omega_tower p) else one.
Lemma omega_tower_nf n: T1nf (omega_tower n).
Lemma omega_tower_unbounded x: ~ (forall n, (omega_tower n) < x).

Definition ex_middle:=
  forall (P: T1 -> Prop),  let Q :=  exists x, (T1nf x /\ P x) in  Q \/ ~Q.

Lemma ex_middle_pick (P: T1 -> Prop):  ex_middle ->
     (exists x, (T1nf x /\ P x))  \/ (forall x, T1nf x -> ~ (P x)).
Lemma min_exists (P: T1 -> Prop) x: ex_middle ->
   T1nf x -> (P x) ->
   exists y, T1nf y /\ P y /\ forall z, T1nf z -> P z -> y <= z.
```

In what follows, we shall avoid the use of the excluded middle principle. This means that we assume a priori the existence of a supremum. This is some ordinal x such that $z \in E$ implies $z \leq x$ and (H): if $z \in E$ implies $z \leq y$, then $x \leq y$. We may also consider (H'): if $y < x$, there is $z \in E$, such that $y < z$; this second assumption is stronger (unless we use classical logic) and will be used hereafter.

We are now faced with the following problem. Let $f(n) = \omega + n$. By normality of addition (see below) the supremum of f is $\omega + \omega$. We show here that the supremum of $[1,0,n]$ is

$[1,0,\omega]$. We expect it however to be $[1,1,0]$. For this reason, we impose y in (H') to be NF. We loose uniqueness of the supremum; however, there is a unique NF supremum, and in this case it is $[1,1,0]$, as expected.

```
Notation Tf := (nat -> T1).

Definition limit_v1 (f: Tf) x :=
    (forall n, f n < x) /\ (forall y, y < x -> (exists n, y <= f n)).
Definition limit_v2 (f: Tf) x :=
    (forall n, f n < x) /\ (forall y, T1nf y -> y < x -> (exists n, y <= f n)).

Lemma limit_unique1 (f: Tf) x x' :limit_v1 f x -> limit_v1 f x' ->
  x = x'.
Lemma limit_unique2 (f: Tf) x x' : limit_v2 f x -> limit_v2 f x' ->
  T1nf x -> T1nf x'->  x = x'.
Lemma limit_CE1: limit_v1 omega_plus_n (cons one 0 T1omega).
Lemma limit_CE2: limit_v2 omega_plus_n (cons one 1 zero).
Lemma limit_CE3: limit_v2 omega_plus_n (cons one 0 T1omega).
```

We say that x is the limit of f if f is strictly increasing, x is NF, and x is the supremum of f (in the second sense). Such a limit is unique.

```
Definition limit_of (f: Tf) x :=
  [/\ (forall n m, (n < m)%N -> f n < f m), limit_v2 f x & T1nf x].

Lemma fincP (f: Tf) :
  (forall n, f n < f n.+1) ->
  (forall n m, (n < m)%N -> f n < f m).

Lemma limit_unique f x y: limit_of f x -> limit_of f y  -> x = y.
Lemma limit_lub f x y: limit_of f x -> (forall n, f n <= y) -> T1nf y ->
  x <= y.
```

Consider the three functions: $\phi_1(a,f) : n \mapsto a + f(n)$, $\phi_2(f) : n \mapsto \phi_0(f(n))$, and $\phi_3 : n \mapsto [b,n,0]$. Assume that the limit of f is b and a is NF. Then the limit of $\phi_1(a,f)$ is $a+b$, and the limit of $\phi_2(f)$ is $\phi_0(b)$. Moreover the limit of ϕ_3 is $\phi_0(a+1)$.

```
Definition phi1 a (f:Tf) := fun n => a + f n.
Definition phi2 (f:Tf) := fun n => phi0 (f n).
Definition phi3 a:= fun n => cons a n zero.

Lemma limit1 a b f: T1nf a -> limit_of f b -> limit_of (phi1 a f) (a + b).
Lemma limit2 b f: limit_of f b -> limit_of (phi2 f) (phi0 b).
Lemma limit3 a: T1nf a -> limit_of (phi3 a) (phi0 (T1succ a)).
```

We construct now, by induction, a function $f(x)$, such the supremum of $f(x)$ is x, whenever x is NF and limit. We take ϕ_3 when $x = 0$ or $x = [a,n,b]$, $a = 0$ (these cases are irrelevant). Let $x_n = [a,n,0]$, so that $x = x_n + b$. Assume first b limit; the solution is $\phi_1(x_n, f(b))$. Otherwise, we have $b = 0$ and $x = x_n$. If a is limit, then $f(x_0) = \phi_2(f(a))$, otherwise $f(x_0) = \phi_3(a-1)$. If $n > 0$, $f(x_n) = \phi_1(x_{n-1}, f(x_0))$.

```
Fixpoint limit_fct x :=
if x is cons a n b then
```

```
  if (b==zero) then
     if(a==zero) then phi3 a
     else if (T1is_succ a)
        then if (n==0) then phi3 (T1pred a) else
         phi1 (cons a n.-1 zero) (phi3 (T1pred a))
     else if(n==0) then (phi2 (limit_fct a))
     else phi1 (cons a n.-1 zero)  (phi2 (limit_fct a))
  else phi1 (cons a n zero) (limit_fct b)
else phi3 zero.

Lemma limit_prop x: T1nf x -> T1limit x -> limit_of (limit_fct x) x.
```

We consider from now on functions $f : \mathrm{T}_1 \to \mathrm{T}1_1$, and define $\sup_{y<x} f(y)$ as the z such that $f(y) \leq z$ whenever $y < x$; moreover, if $z' < z$ there is y such that $y < x$ and $z' < f(y)$. As above, we assume z and z' NF, but also assume y NF. We say that the function f is *normal* if it is strictly increasing, and $f(x) = \sup_{y<x} f(y)$ whenever x is limit. We shall consider only NF arguments, and assume $f(x)$ NF when x is NF.

Let f be the identity function; in this case $\sup_{y<x} f(y)$ is x when x is zero or limit, it is y when x is the successor of y.

```
Definition sup (f: T1-> T1) x z :=
  [/\ T1nf z,
      (forall y, T1nf y -> y < x -> f y <= z) &
      (forall z', T1nf z' -> z' < z -> exists y,
          [&& T1nf y, y < x & z' < f y])].
Definition normal f:=
  [/\ forall x, T1nf x -> T1nf (f x),
      (forall x y, T1nf x -> T1nf y -> x < y -> f x < f y)&
      (forall x, T1nf x -> T1limit x -> sup f x (f x)) ].

Lemma sup_unique f x z z': sup f x z -> sup f x z' -> z = z'.
Lemma sup_0alpha_zero: sup id zero zero.
Lemma sup_0alpha_limit x: T1nf x -> T1limit x -> sup id x x.
Lemma sup_0alpha_succ x: T1nf x -> sup id (T1succ x) x.
Lemma normal_id: normal id.
Lemma normal_limit f x: normal f -> T1nf x -> T1limit x -> T1limit (f x).
Lemma normal_compose f g: normal f -> normal g -> normal (f \o g).
```

2.6 Multiplication

Multiplication is defined by induction on the second argument as follows.

```
Fixpoint T1mul (c1 c2 : T1) {struct c2}:T1 :=
  if c2 is cons a' n' b' then
    if c1 is cons a n b then
       if((a==zero) && (a' == zero)) then cons zero (n*n' + n + n')%N b'
       else if(a'==zero) then  cons a  (n*n' + n + n')%N b
       else cons (a + a') n' ((cons a n b) * b')
     else zero
  else zero
where  "c1 * c2" := (T1mul c1 c2) : cantor_scope.
```

If one argument is zero, so is the product, and conversely. Note that if $x = \omega^a \cdot n + b$, if $a = 0$, the product $x \cdot y$ is independent of b. In particular, $x \cdot 1$ is not always x. We have $\phi_0(a) \cdot \phi_0(b) = \phi_0(a+b)$.

```
Lemma T1muln0 x:  x * zero = zero.
Lemma T1mul0n x:  zero * x  = zero.
Lemma T1mul_eq0 x y: (x * y == zero) = (x== zero) || (y == zero).
Lemma T1mul_eq1 a b: T1nf a -> (a* b == one) = ((a == one) && (b == one)).
Lemma mul_na n b x:  (cons zero n b) * x =  (cons zero n zero) * x.
Lemma mul_int n m : \F n * \F m = \F (n *m)%N.
Lemma mul_phi0 a b: phi0 (a + b) = phi0 a * phi0 b.
Lemma T1mul1n x: one * x  = x.
Lemma T1muln1 x: T1nf x -> x * one = x.

Lemma T1mul1nCE (x := T1bad): x * one  <> x.
```

Let's state two non-trivial properties: $x \cdot (y+z) = (x \cdot y) + (x \cdot z)$ $x \cdot (y \cdot z) = ((x \cdot y) \cdot z)$,.

```
Lemma mul_distr: right_distributive T1mul T1add.
Lemma mulA: associative T1mul.
```

If $b < \phi_0(x)$ then $n \cdot b < \phi_0(x)$ (provided x non-zero). Thus multiplication is normal. Multiplication is also increasing. As $n \cdot \omega = \omega$, for any non-zero integer n, we cannot expect multiplication to be strictly increasing in its first argument.

```
Lemma ltn_simpl2 n n' n'':
   (n * n' + n + n' < n * n'' + n + n'')%N =  (n' < n'')%N.
Lemma eqn_simpl2 n n' n'':
   (n * n' + n + n' == n * n'' + n + n'')%N =  (n' == n'')%N.

Lemma T1lt_mul2l x y z: x != zero -> T1nf z -> ((x *y < x *z) = (y < z)).
Lemma T1lt_mul2r x y z:  (y * x < z * x) ->  (y < z).
Lemma T1le_mul2l x y z :  x != zero -> T1nf y ->
  (x *y <= x *z) = (y <= z).
Lemma T1le_mul2r x y z:  (y <= z) -> (y * x <= z * x).
Lemma nf_mul a b: T1nf a -> T1nf b -> T1nf (a * b).
Lemma T1eq_mul2l p m n : p != zero -> T1nf m -> T1nf n ->
   (p * m == p * n) = (m == n).
Lemma T1le_pmulr x a: T1nf a -> x != zero -> a <= a * x.
Lemma T1le_pmull x a: x != zero -> a <= x * a.
Lemma T1le_pmulrl x a: x != zero -> a <= x * a.
Lemma T1le_mul m1 m2 n1 n2 : T1nf m2 -> m1 <= n1 -> m2 <= n2 ->
   m1 * m2 <= n1 * n2.

Lemma mult_fin_omega n: (\F (S n)) * T1omega = T1omega.
Lemma T1lt_mul2lCE (x := one)(y := one ) (z:= T1bad):
  ((y < z) != (x *y < x *z)).
Lemma T1le_pmulrCE (x:= \F1 ) (a:=T1bad) : (a <= a * x) = false.
Lemma T1le_mulCE (m1:= one) (m2:= T1bad) (n1 := \F1) (n2 := one) :
   (m1 <= n1) && (m2 <= n2) && ( m1 * m2 <= n1 * n2) == false.
```

If y is non-zero, then $\omega \cdot y$ is a limit ordinal, and conversely. In particular, we can always uniquely write $x = \omega \cdot y + n$, where n is an integer (existence requires x NF, and y is then NF).

```
Fixpoint T1div_by_omega x :=
  if x is cons a n b then cons (a - one) n (T1div_by_omega b) else zero.

Lemma T1mul_omega a n b:
  T1omega * (cons a n b) =
  if (a== zero) then cons (one) n zero else cons (one + a) n (T1omega * b).

Lemma mul_int_limit n y: T1limit y -> \F n.+1 * y = y.
Lemma mul_omega_limit x: x != zero -> T1limit (T1omega * x).
Lemma div_by_omega_pr x: T1nf x -> ((x==zero) || T1limit x)
  -> T1omega * (T1div_by_omega x) = x.
Lemma nf_div_by_omega x: T1limit x -> T1nf x -> T1nf (T1div_by_omega x).
Lemma nf_rev x (u := (T1div_by_omega (T1split x).1)) (v:= (T1split x).2):
  T1nf x ->  T1nf u /\  x = T1omega * u + \F v.
Lemma nf_rev_unique u v (x:= T1omega *u + \F v): T1nf u ->
    u = T1div_by_omega (T1split x).1 /\ v = (T1split x).2.
Lemma nf_revCE u v: T1bad <> T1omega * u  + \F v.
```

As noted above $x \cdot 1$ may be different from x. This can only happen when $x = [a, n, b]$, a is zero, b is non-zero. We have $a + a \cdot \omega \cdot b = a \cdot \omega \cdot b$ whenever b is non-zero (this is essentially $1+\omega = \omega$, no argument needs to be NF). If n is is an integer, $n+\omega \cdot b = \omega \cdot b$. Assume $x = \omega \cdot u + n$, and $y = \omega \cdot v + m$. We give the corresponding formula for $x + y$ and $x \cdot y$.

```
Lemma T1muln1_CE x:
  (x == x * one) =
    (if x is cons a n b then ((a != zero) || (b== zero)) else true).

Lemma T1muln1_CE x: (x = x * one) <-> ((exists n, x = \F n) \/ ~~T1finite x).
Lemma add_simpl3 x y: y != zero ->
  x + x * (T1omega * y) = x * (T1omega * y).
Lemma plus_int_Ox n x: x != zero -> \F n + T1omega * x =  T1omega * x.
Lemma mul_sum_omega a n: a != zero ->
   (T1omega * a + \F n) * T1omega  = (T1omega * a) * T1omega.

Lemma nf_rev_sum x y
  (u := T1div_by_omega (T1split x).1) (n:= (T1split x).2)
  (v := T1div_by_omega (T1split y).1) (m:= (T1split y).2)
  (w := T1div_by_omega (T1split (x+y)).1) (p:= (T1split (x+y)).2):
  T1nf x -> T1nf y ->
  if (v==zero) then (w = u /\ p = (n + m)%N) else (w = u+v /\ p = m).

Lemma nf_rev_prod x y
  (u := T1div_by_omega (T1split x).1) (n:= (T1split x).2)
  (v := T1div_by_omega (T1split y).1) (m:= (T1split y).2)
  (w := T1div_by_omega (T1split (x*y)).1) (p:= (T1split (x*y)).2):
  T1nf x -> T1nf y ->
  if (u== zero)
     then if  (n == 0) then (w = zero /\ p = 0)
     else  (w = v /\ p = (n*m)%N)
  else if (m==0) then (w = u * T1omega *v  /\ p = 0)
  else  (w = u * T1omega *v + u * \F m /\ p = n).
```

Let's show that multiplication is normal. We must show that $\sup_{x<b} a \cdot x = a \cdot b$, whenever b is a limit ordinal. The non-trivial point is to show H(b): if $z < a \cdot b$, there is y such that $y < b$ and $z < a \cdot y$. Consider the case $b = \omega$ to start with. Let $a = [a_1, n_1, b_1]$, and $z = [a_2, n_2, b_2]$. We

have $a \cdot b = [a_1 + 1, 0, 0]$, and the condition $z < a \cdot b$ becomes $a_2 \leq a_1$. If $a_2 < a_1$, we take $y = 1$, otherwise $y = n_2 + 2$. If b is non-zero then $\mathrm{H}(b)$ implies $\mathrm{H}(c + b)$.

```
Lemma mul_omega_pr1 a: a != zero -> T1nf a ->
  sup (T1mul a) T1omega (a * T1omega).
Lemma mul_omega2_pr1 a (u:= cons one 1 zero): a != zero -> T1nf a ->
  sup (T1mul a) u (a * u).
Lemma mul_omega_pr3 a b c: a != zero -> c != zero ->
   T1nf a -> T1nf b -> T1nf c ->
  sup (T1mul a) c (a * c) ->
  sup (T1mul a) (b+c) (a * (b + c)).
```

(proof still missing)

2.7 Ordinal Power

The original definition of exponential by Castéran was:

```
(*
Fixpoint exp  (a b : T1) {struct b}:T1 :=
  match a,b with
 |  x, zero => cons zero 0 zero
 | cons zero 0 _ , _ => cons zero 0 zero
 | zero, y            => zero
 | x , cons zero n' _ =>  exp_F x (S n')
 | cons zero n _, cons  (cons zero 0 zero) n'  b' =>
       ((cons (cons zero n' zero) 0 zero) *
               ((cons zero n zero) ^  b'))
 | cons zero n _, cons  a' n' b' =>
           (omega_term
                   (omega_term (a' - (F 1)) n')
                   0) *
                ((cons zero n zero) ^ b')
 | cons a  n b, cons  a' n' b' =>
    ((omega_term   (a * (cons a' n' zero))
                           0) *
           ((cons a n b) ^ b'))
end
where "a ^ b" := (exp a b) : cantor_scope.
*)
```

We shall use here a different approach: assume that equations (2.1) and (2.2) hold. We know that every y can be written in the form $y = \omega \cdot a + n$, so that $x^y = (x^\omega)^a \cdot x^n$. The quantity x^n is computed by repeated multiplications, as in the case of Castéran; the quantity x^ω is x when x is zero or one, ω in the case, x finite at least two, and $\omega^{u\cdot\omega}$ when $x = [u, n, v]$ is infinite. So, unless x is zero, x^ω has the form ω^c and $(x^\omega)^a = \omega^{c\cdot a}$ (see [3], theorem 19B). Finally, we use $\omega^x = \phi_0(x)$. Thus, we define $x^y = \mathrm{P_O}(x, a) \cdot \mathrm{P_F}(x, n)$.

```
Fixpoint exp_F a n :=
  if n is p.+1 then a * (exp_F a p) else one.

Definition exp_O a b :=
  if (a==zero) then if (b== zero) then one else a
```

```
  else if (a== one) then one
  else if (T1finite a) then (phi0 b)
  else phi0 ((T1log a) * T1omega * b).

Definition T1exp a b:=
  (exp_O a (T1div_by_omega (T1split b).1)) * (exp_F a ( (T1split b).2)).

Notation "a ^ b" := (T1exp a b) : cantor_scope.
```

We study here the function P_O.

```
Lemma nf_expO a b: T1nf a -> T1nf b -> T1nf (exp_O a b).
Lemma expO_n0 x: exp_O x zero = one.
Lemma expO_1n n: exp_O (one) n  = one.
Lemma expO_eq0 a b: (exp_O a b == zero) = ((a== zero) && (b != zero)).
Lemma expO_eq1 a b: (exp_O a b == one) = ((a== one) || (b == zero)).
Lemma expO_add z u v: exp_O z u * exp_O z v = exp_O z (u + v).
```

We study here the function P_F.

```
Lemma nf_expF a n: T1nf a -> T1nf (exp_F a n).
Lemma expF_1n n: exp_F (one) n  = one.
Lemma expF_eq0 a n: (exp_F a n == zero) = ((a== zero) && (n != 0)).
Lemma expF_eq1 a n: T1nf a -> (exp_F a n == one) = ((a== one) || (n == 0)).
Lemma expF_add a n m: (exp_F a n) * (exp_F a m) = exp_F a (n + m).
Lemma expF_mul a n m: exp_F a (n * m) =  exp_F (exp_F a n) m.
```

We deduce the following properties.

```
Lemma nf_exp a b: T1nf a -> T1nf b -> T1nf (a ^b).
Lemma expx_pnat x n b: x ^ (cons zero n b)  = exp_F x n.+1.
Lemma expx_nat x n:  x ^ \F n = exp_F x n.
Lemma exp00: zero ^zero = one.
Lemma expx0 x: x ^zero = one.
Lemma expx1 x: T1nf x -> x ^ one = x.
Lemma exp1x x: one ^ x = one.
Lemma exp_int a b: (\F a) ^ (\F b) = \F (a ^b%N).
Lemma exp_eq0 x y:  x^y ==  zero = ((x==zero) && (y != zero)).
Lemma exp_eq1 x y: T1nf x -> T1nf y ->
Lemma exp0nz x: x != zero -> zero ^ x =  zero.
  (x^y == one) = ((x== one) || (y == zero)).
Lemma exp2omega n: (\F n.+2)^ T1omega = T1omega.
Lemma expx1CE: T1bad ^ one = one.
```

If $x = [a,n,b]$ is NF, then $x = \omega^a \cdot (n+1) + b$. This is the Cantor Normal Form, (1.4), since x NF implies $b < \phi_0(a)$. We have also uniqueness.

```
Lemma pow_omega x: T1nf x -> T1omega ^x = phi0 x.

Lemma cantor_exists a n b: T1nf (cons a n b) ->
    cons a n b = (T1omega^a) * (\F n.+1) + b.
Lemma cantor_unique a n b a' n' b':
  T1nf (cons a n b) -> T1nf (cons a' n' b') ->
  (T1omega^a) * (\F n.+1) + b = (T1omega^a') * (\F n'.+1) + b' ->
```

```
  (a=a' /\ n = n' /\ b = b').

Lemma cantor_CE1 : T1omega ^ T1bad != phi0 T1bad.
Lemma cantorCE2: cons zero 0 T1omega != (T1omega^ zero) * (one) + T1omega.
Lemma cantorCE3: cons T1bad 0 zero  != (T1omega^ T1bad) * (one) + zero.
```

Let's state some properties of the logarithm. We have $\log(x\cdot y)=\log(x)\cdot\log(y)$, so that $\log(x^n)=\log(x)\cdot n$, whenever n is an integer. Let $y=\omega\cdot u+n$. It follows that $\log(x^y)=\log(\mathrm{P_O}(x,u))+\log(x^n)$. Assume first that x is a non-trivial integer. Then $\log(x^y)=u$. On the other hand, if x is infinite, then $\log(\mathrm{P_O}(x,u))=(\log x)\cdot\omega\cdot u$, so that $\log(x^y)=(\log x)\cdot y$.

```
Lemma T1log_prod a b: a != zero -> b != zero ->
    T1log(a * b) = T1log a + T1log b.
Lemma T1log_exp0 x n: T1nf x -> T1log (exp_F x n) = (T1log x) * (\F n).
Lemma T1log_exp1 z x: T1nf z -> T1nf x -> ~~ T1finite z ->
   T1log (z ^ x) = (T1log z) * x.
Lemma T1log_exp2 z u v: (z == zero) = false -> (z == one) = false ->
   T1finite z -> T1nf u -> T1log (z ^ (T1omega * u + \F v)) = u.
```

Let's show

$$z^{x+y}=z^x\cdot z^y, \tag{2.1}$$

$$z^{x\cdot y}=(z^x)^y. \tag{2.2}$$

We start with

$$\mathrm{P_F}(z,n)\cdot\mathrm{P_O}(z,v)=\mathrm{P_O}(z,v). \tag{2.3}$$

The proof is by induction on n, and the non-trivial case is when $\mathrm{P_F}(z,n)=z$. We deduce

$$\mathrm{P_O}(z,z\cdot n)\cdot z^m=(\mathrm{P_O}(z,v)\cdot z^m)^n \qquad (n>0) \tag{2.4}$$

(let $u=\mathrm{P_O}(z,v)$ and $v=z^m$; the RHS has the form $uvuv\cdots uv$, and we can remove each occurence of v but the last). We have

$$\mathrm{P_O}(z^x,u)=\phi_0(\log(z^x)\cdot\omega\cdot u) \tag{2.5}$$

when x is infinite and z non-zero (since z^x is infinite when z is not one). We also have

$$\mathrm{P_O}(z^n,a)=\mathrm{P_O}(z,a). \tag{2.6}$$

Proof of (2.1). Write $x=\omega\cdot u+n$, $y=\omega\cdot v+m$. If $v=0$, we have $x+y=\omega\cdot u+(n+m)$. This is the trivial case. Otherwise $x+y=\omega\cdot(u+v)+m$, and the result follows from (2.3).

Proof of (2.2). The case where x is finite follows from (2.6). The case $z=0$ or $z=1$ are trivial. Assume that x and y are as above. The RHS of (2.2) is $(z^x)^y=\mathrm{P_O}(z^x,v)\cdot(z^x)^m$. If m is non-zero we use (2.4), and z^n appears as a factor on the right. This solves the case where y is finite. Otherwise $x\cdot y=\omega\cdot u\cdot(\omega\cdot v+m)+n$, say $x\cdot y=\omega\cdot\alpha+n$. We can simplify by z^m, and are reduced to show $\mathrm{P_O}(z,\alpha)=\mathrm{P_O}(z^x,v)\cdot\mathrm{P_O}(z,u\cdot m)$. If z is finite, the LHS is $\phi_0(\alpha)$, otherwise it is $\phi_0(\log(z)\cdot\omega\cdot\alpha)$. The second factor of the RHS can be similarly evaluated. We use (2.5) for the first factor. The result follows.

```
Lemma exp_FO z n v: v != zero -> exp_F z n * exp_O z v = exp_O z v.
Lemma exp_FO1 z v n m: T1nf z -> T1nf v -> v != zero -> n != 0 ->
  exp_O z (v * \F n) * exp_F z m = exp_F (exp_O z v * exp_F z m) n.
Lemma exp_FO2 z m u: T1nf z -> m != 0 -> exp_O (exp_F z m) u = exp_O z u.
Lemma exp_FO3 z x u (w := T1div_by_omega (T1split x).1):
   T1nf z -> T1nf w -> (w == zero) = false -> (z == zero) = false ->
  exp_O (z ^ x) u = phi0( T1log (z ^x) * T1omega * u).

Lemma exp_sum x y z: T1nf x -> T1nf y -> z ^(x+y)  = z ^x * z ^y.
Lemma exp_prod x y z: T1nf z -> T1nf x -> T1nf y -> z ^(x *y)  = (z ^x) ^y.
```

We show here some monotonicity properties.

```
Lemma pow_mon1 x y z: T1nf x -> T1nf y -> T1nf z -> x != zero ->
   y <= z -> x ^y <= x ^z.
Lemma pow_mon2 x y z: T1nf x -> T1nf y -> T1nf z -> x != zero ->  x != one ->
   y < z -> x ^y < x ^z.
Lemma pow_mon3 x y z: T1nf x -> x <= y -> x ^z <= y ^z.
```

Chapter 3

The second model

3.1 Introduction

Let $\phi_a(b)$ be the function defined by transfinite induction on von Neumann ordinals as follows: $\phi_0(x) = \omega^x$, and $\phi_a(x)$ enumerates, in order, all ordinals t such that $\phi_b(t) = t$, whatever $b < a$. For any a, $\phi_a(x)$ is normal. Let's define ψ_a as follows: If b has the form $b_0 + n$, where n is an integer, and $\phi_a(b_0) = b_0$, then $\psi_a(b) = \phi_a(b+1)$, otherwise $\psi_a(b) = \phi_a(b)$. This function is strictly increasing, and has the same range as ϕ_a. For any x, there is a unique pair (a, b) such that $\omega^x = \psi_a(b)$. The function ϕ satisfies the following property:

$$\phi_a(b) < \phi_c(d) \iff \begin{cases} \text{if } a < c \text{ then } b < \phi_c(d) \\ \text{if } a = c \text{ then } b < d \\ \text{if } c < a \text{ then } \phi_a(b) < d. \end{cases} \tag{3.1}$$

We deduce (note the ≤ on the last clause):

$$\psi_a(b) < \psi_c(d) \iff \begin{cases} \text{if } a < c \text{ then } b < \psi_c(d) \\ \text{if } a = c \text{ then } b < d \\ \text{if } c < a \text{ then } \psi_a(b) \le d. \end{cases} \tag{3.2}$$

Let z be any non-zero ordinal; in its Cantor Normal Form (1.4), we replace ω^x by $\psi_a(b)$, and obtain

$$x = \psi_a(b) \cdot (n+1) + c, \qquad c < \psi_a(b). \tag{3.3}$$

Let Γ_0 be the least ordinal a such that $\phi_a(0) = a$. If $x = \psi_a(b) < \Gamma_0$, then $a < x$ and $b < x$. Each of a, b, c is zero or has a form (3.3), case where we can replace a, b or c; this recursive procedure will eventually stop.

What we get is then an instance of the structure T_2 below; not all members are obtained because of the condition $c < \psi_a(b)$. Equation (3.3) will induce an ordering on T_2. Elements of type T_2 will be called "ordinals"; one can associate a von Neumann ordinal $O(x)$ via if x is zero, then $O(x)$ is the ordinal zero, if x is '`cons a b n c`', then $O(x) = \psi_{O(a)}(O(b)) \cdot (n+1) + O(c)$; see the last chapter of this report.

We shall not define multiplication here, since there is no explicit formula that gives $\phi_a(b) \cdot \phi_c(d)$ in terms of ϕ (assume $\phi_a(b) = \omega^u$, $\phi_c(d) = \omega^v$, and the product is $\phi_x(y) = \omega^w$. Obviously $w = u + v$, but computing x and y from w is not possible).

We thus introduce the following type T_2 and equip it with a boolean equality.

```
Inductive T2 : Set :=
  zero : T2
| cons : T2 -> T2  -> nat -> T2 -> T2.

Fixpoint T2eq x y {struct x} :=
  match x, y with
  | zero, zero  => true
  | cons a b n c, cons a' b' n' c' =>
      [&& T2eq a a', T2eq b b',  n== n' & T2eq c c' ]
  | _, _ => false
end.

Lemma T2eqP : Equality.axiom T2eq.

Lemma T2eqE a b n d a' b' n' d':
  (cons a b n d == cons a' b' n' d') =
      [&& a == a', b == b', n== n' & d == d' ].
```

We introduce some definitions and notations. We shall write $[a,b,n,c]$ instead of '`cons a b n c`', and $[a,b]$ instead of $[a,b,0,0]$. This represents $\psi_a(b)$. We consider an injection $\mathbf{N} \to \mathrm{T}_2$ and an injection $\mathrm{T}_1 \to \mathrm{T}_2$.

```
Notation "[ x , y ]" := (cons x y 0 zero) (at level 0) :g0_scope.

Definition one := [zero, zero].
Definition omega  := [zero, one].
Definition epsilon0  := [one,zero].

Definition T2finite x:=
  if x is cons a b n c then ((a==zero) && (b== zero)) else true.
Definition psi a b  := [a, b].


Definition T2nat p := if p is n.+1 then cons zero zero n zero else zero.
Fixpoint T1T2 (c:T1) : T2 :=
  if c is CantorOrdinal.cons a n b then cons zero (T1T2 a) n (T1T2 b)
  else zero.
Notation "\F n" := (T2nat n)(at level 29) : g0_scope.
```

3.2 Comparison

Comparison of two ordinals $x < y$, of the form $\psi_a(b)$ is given by (3.2); we cannot define it by induction on the first nor the second argument, as both x and y appear on the RHS. However, it is possible to proceed by induction on the maximum of the "length" of the arguments.

Castéran uses the length defined in [7] by the following rule. Write $x = \sum_{i<\mathrm{N}} \psi_{a_i}(b_i)$. Then $l(x)$ is the sum of N and twice the maximum of the $l(a_i)$, $l(b_i)$. The quantity N is defined by $\mathrm{N}([a,b,n,c]) = n+1+\mathrm{N}(c)$ and $l(x) = \mathrm{N}(x) + 2l'(x)$, where $l'([a,b,n,c])$ is the maximum of $l(a)$, $l(b)$ and $l'(c)$.

This definition is too complicated. We use here the size $s(x)$, defined by $s(0) = 0$ and $s([a,b,n,c]) = 1+\max(s(a),s(b),s(c))$. Our proofs, in the case of two arguments, will be by induction on $l(x,y) = s(x)+s(y)+1$.

If $x = [a,b,n,c]$, then $s(a) < s(x)$, $s(b) < s(x)$, $s(c) < s(x)$ and $s([a,b]) \leq s(x)$.

```
Fixpoint size x :=
  if x is cons a b n c then
     (maxn (size a) (maxn (size b) (size c))).+1
  else 0.

Lemma size_prop1 a b n c (l:= size (cons a b n c)):
   [/\ size a < l, size b < l, size c < l & size [a, b] <= l]%N.

Lemma size_prop a b n c a' b' n' c'
   (l := size (cons a b n c) + size (cons a' b' n' c')) :
   (size c + size c' < l)%N /\ (size [a, b] + size b' < l)%N /\
   (size a' + size a < l)%N /\ (size b + size b' < l)%N /\
   (size b + size [a', b'] < l)%N /\ (size a + size a' < l)%N.
```

Let $f(x,y)$ be the function that compares two ordinals. It satisfies a given equality, that follows from (3.2), say $f(x,y) = \mathrm{C}(f,x,y)$. We first define, by induction on k, an auxiliary function f_k satisfying $f_{k+1}(x,y) = \mathrm{C}(f_k,x,y)$, then consider $f(x,y) = f_{l(x,y)}(x,y)$.

```
Definition lt_rec f x y :=
 if x is cons a b n c then
   if y is cons a' b' n' c' then
     if (  ((f a a')  && (f b ([a', b'])))
         || ((a == a') && (f b b'))
         || ((f a' a)  && (f ([a, b]) b'))
         || (((f a' a)  && ([a, b] == b'))))
     then true
     else if ((a== a') && (b==b')) then
       if (n < n')%N then true
       else if (n == n') then  (f c c') else false
       else false
   else false
 else if y is cons a' b' n' c' then true else false.

Fixpoint T2lta k {struct k}:=
 if k is k.+1 then lt_rec (T2lta k) else fun x y => false.

Definition T2lt a b :=  T2lta ((size a) + size b).+1 a b.
Definition T2le (x y :T2) := (x == y) || (T2lt x y).
Notation "x < y" := (T2lt x y) : g0_scope.
Notation "x <= y" := (T2le x y) : g0_scope.
Notation "x >= y" := (y <= x) (only parsing) : g0_scope.
Notation "x > y"  := (y < x) (only parsing)  : g0_scope.
```

By induction, if k is big enough, f_k is independent of k. It follows $f(x,y) = \mathrm{C}(f,x,y)$. As a byproduct (3.2) holds.

```
Definition lt_psi a b a' b':=
      ((a < a')  && (b < [a', b']))
   || ((a == a') && (b < b'))
   || ((a' < a)  && ([a, b] < b'))
   || ((a' < a)  && ([a, b] == b')).

Lemma T2ltE x y : x <y  = lt_rec T2lt x y.
Lemma T2lt_psi a b a' b': [a,b] < [a', b'] = lt_psi a b a' b'.
Lemma T2lt_consE a b n c a' b' n' c' :
```

```
   cons a b n c < cons a' b' n' c' =
     if (lt_psi a b a' b') then true
     else if ((a== a') && (b==b')) then
       if (n < n')%N then true
       else if (n == n') then  (c < c') else false
       else false.
Lemma T2le_consE a b n c a' b' n' c' :
   cons a b n c <= cons a' b' n' c' =
     if (lt_psi a b a' b') then true
     else if ((a== a') && (b==b')) then
       if (n < n')%N then true
       else if (n == n') then  (c <= c') else false
       else false.
Lemma T2lt_psi_aux a b a' b': a < a' -> b < [a', b'] -> [a,b] < [a',b'].
```

We prove the same results as in the case T1. Note that the mapping $T_1 \to T_2$ is strictly increasing.

```
Lemma T2ltn0 x: (x < zero) = false.
Lemma T2lt0n x: (zero < x) = (x != zero).
Lemma T2le0n x: zero <= x.
Lemma T2len0 x: (x <= zero) = (x == zero).
Lemma omega_lt_epsilon0: omega < epsilon0.

Lemma T2ltnn x: (x < x) = false.
Lemma T2lt_ne a b : a < b -> (a == b) = false.
Lemma T2lt_ne' a b : a < b -> (b == a) = false.
Lemma T2ltW a b : (a < b) -> (a <= b).
Lemma T2le_eqVlt a b : (a <= b) = (a == b) || (a < b).
Lemma T2lt_neAle a b : (a < b) = (a != b) && (a <= b).
Lemma T2ge1 x:  (one <= x) = (x != zero).
Lemma T2lenn x: x <= x.
Lemma T2lt1 x: (x < one) = (x==zero).
Lemma psi_lt1 a b c  n a' b':
  cons a b n c < [a', b'] = ([a, b] < [a', b']).
Lemma psi_lt2 a b n c n' c':  cons a b n' c' < cons a b n c =
 (if (n' < n)%N then true else if n' == n then c' < c else false).
Lemma T2nat_inc n p : (n < p)%N = (\F n < \F p).
Lemma T1T2_inj n p : (n == p) = (T1T2 n == T1T2 p).
Lemma T1T2_inc n p : (n < p)%ca = (T1T2 n < T1T2 p)%g0.
```

We show by induction, as explained above, that one of $x < y$, $y < x$ has to be false, and if $x \neq y$, one has to be true. The lemmas `T2leP`, `T2ltP` and `T2ltgtP` are similar to those given for T_1.

```
Lemma T2lt_anti b a: a < b -> (b < a) = false.
Lemma T2lt_trichotomy a b: [|| (a< b), (a==b) | (b < a)].
Lemma T2leNgt a b: (a <= b) = ~~ (b < a).
Lemma T2ltNge a b: (a < b) = ~~ (b <= a).
Lemma T2eq_le m n : (m == n) = ((m <= n) && (n <= m)).
Lemma T2leP x y : T2leq_xor_gtn x y (x <= y) (y < x).
Lemma T2ltP m n : T2ltn_xor_geq m n (n <= m) (m < n).
Lemma T2ltgtP m n : compare_T2 m n (m < n) (n < m) (m == n).
Lemma T2gt1 x: (one < x) = ((x != zero) && (x != one)).
```

We show transitivity by induction on n, such that $l(a) + l(b) + l(c) < n$. The trick is that, if

$[a,b,n,c] < [a',b',n',c']$, then in most of the cases we have $[a,b] < [a',b']$, and these relations might simplify, for instance as $[a,b] < b'$, so that, if $b' < b''$, we have $[a,b] < b''$ by induction.

We have $b < \psi_a(b)$ and $a < \psi_a(b)$.

```
Lemma T2lt_trans b a c: a < b -> b < c -> a < c.
Lemma T2lt_le_trans b a c: a < b -> b <= c -> a < c.
Lemma T2le_lt_trans b a c: a <= b -> b < c -> a < c.
Lemma T2le_trans b a c: a <= b -> b <= c -> a <= c.

Lemma T2le_psi1 a b n c: [a, b] <= cons a b n c.
Lemma T2lt_psi_b a b: b < [a,b].
Lemma T2lt_psi_a a b: a < [a,b].
```

3.3 Normal Form

We say that x is NF if $x = [a,b,n,c]$ and $c < \psi_a(b)$. We give here simple properties. Note that if x is NF, then $x < \omega$ says that x is the image of the injection $\mathbf{N} \to T_2$, and $x < \epsilon_0$ says that x is the image of the injection $T_1 \to T_2$.

```
Fixpoint T2nf x :=
  if x is cons a b n c then [&& T2nf a, T2nf b, T2nf c & c < [a,b] ]
  else true.

Lemma T2nf_cons_cons a b n a' b' n' c':
  T2nf(cons a b n (cons a' b' n' c')) =
   [&& [a', b'] < [a, b], T2nf a, T2nf b & T2nf(cons a' b' n' c') ].
Lemma nf_psi a b n: T2nf (cons a b n zero) = T2nf a && T2nf b.
Lemma nf_omega : T2nf omega.
Lemma nf_one : T2nf one.
Lemma nf_finite n: T2nf (\F n).
Lemma lt_tail a b n c: T2nf (cons a b n c) ->  c < cons a b n c.
Lemma T1T2range1 x: T1T2 x < epsilon0.
Lemma T1T2range2 x: T2nf x -> x < epsilon0 -> {y: T1 | x = T1T2 y}.
```

3.4 Successor

We define $F(x)$, meaning x is finite, $L(x)$, meaning x is limit, $S(x)$ meaning x is a successor, $s(x)$, the successor of x, $p(x)$ the predecessor of x, and $l(x), t(x)$ the limit part of x and the last coefficient; these two quantities are returned by `T2split`.

If $x = 0$, then $F(x)$ holds, $S(x)$ and $L(x)$ are false. We have $s = 1$, $p = 0$, $l = 0$ and $t = 0$. If $x = [a,b,n,c]$ and $a = b = 0$ (this condition is equivalent to $[a,b] = 1$), then x NF says $c = 0$; in the definition that follows we shall ignore c. In this case F is true, S is true and L is false. $s = n+2$, $p = n$, $l = 0$, $t = n+1$.

Otherwise F is false, L is $c = 0$ or $L(c)$ and S is $S(c)$. Moreover $s = [a,b,n,s(c)]$, $p = [a,b,n,p(c)]$, $l = [a,b,n,l(c)]$, $t = t(c)$,

```
Definition T2finite x:=
  if x is cons a b n c then ([a,b]==one) else true.

Fixpoint T2limit x :=
```

```
  if x is cons a b n c then
    if ([a,b]==one) then false else (c== zero) || T2limit c
  else false.

Fixpoint T2split x:=
 if x is cons a b n c then
      if ([a,b]==one) then (zero, n.+1) else
     let: (x, y) := T2split c in (cons a b n x,y)
   else (zero,0).

Fixpoint T2is_succ x :=
  if x is cons a b n c then ([a,b]==one) || T2is_succ c else false.

Fixpoint T2succ x :=
  if x is cons a b n c
     then if ([a,b]==one) then \F n.+2  else cons a b n (T2succ c)
  else one.

Fixpoint T2pred x :=
  if x is cons a b n c then
     if ([a,b]==one) then \F n else (cons a b n (T2pred c))
  else zero.
```

We show here some properties.

```
Lemma T2finite1 n: T2finite (\F n).
Lemma T2nf_finite a b n c: [a,b]==one -> T2nf (cons a b n c) -> c = zero.
Lemma split_finite x: ((T2split x).1 == zero) = T2finite x.
Lemma T2finite2 x: T2finite x -> T2nf x -> x = \F ((T2split x).2).
Lemma omega_least_inf1 x: T2finite x -> x < omega.
Lemma omega_least_inf2 x: ~~ T2finite x -> omega <= x.
Lemma split_limit x: ((T2split x).2 == 0) = ((x==zero) || T2limit x).
```

More properties.

```
Lemma split_is_succ x: ((T2split x).2 != 0) = (T2is_succ x).
Lemma split_succ x: let:(y,n):= T2split x in T2split (T2succ x) = (y,n.+1).
Lemma split_pred x: let:(y,n):= T2split x in T2split (T2pred x) = (y,n.-1).
Lemma split_le x :  (T2split x).1 <= x.
Lemma nf_split x : T2nf x -> T2nf (T2split x).1.
Lemma T2finite_succ x: T2finite x -> T2finite (T2succ x).
Lemma T1succ_nat n: T2succ (\F n) = \F (n.+1).
Lemma limit_pr1 x: (x == zero) (+) (T2limit x (+) T2is_succ x).
Lemma limit_pr x y: T2limit x -> y < x -> T2succ y < x.
Lemma pred_le a: T2pred a <= a.
Lemma T2le_psi_b a b : T2succ b <= [a,b].
Lemma pred_lt a: T2is_succ a -> T2pred a < a.
Lemma succ_lt a: a < T2succ a.
Lemma nf_succ a: T2nf a -> T2nf (T2succ a).
Lemma nf_pred a: T2nf a -> T2nf (T2pred a).
Lemma succ_pred x: T2nf x -> T2is_succ x -> x = T2succ (T2pred x).
Lemma succ_p1 x: T2is_succ (T2succ x).
Lemma pred_succ x: T2nf x -> T2pred (T2succ x) = x.
Lemma succ_inj x y: T2nf x -> T2nf y -> (T2succ x == T2succ y) = (x==y).
Lemma lt_succ_succ x y: T2succ x < T2succ y -> x < y.
Lemma le_succ_succ x y: x <= y -> T2succ x <= T2succ y.
```

```
Lemma lt_succ_succE x y: T2nf x -> T2nf y ->  (T2succ x < T2succ y) = (x < y).
Lemma le_succ_succE x y:
  T2nf x -> T2nf y -> (T2succ x <= T2succ y) = (x <= y).
Lemma lt_succ_le_1 a b : T2succ a <= b ->  a < b.
Lemma lt_succ_le_2 a b:  T2nf a -> a < T2succ b ->  a <= b.
Lemma lt_succ_le_3 a b:  T2nf a -> (a < T2succ b) = (a <= b).
Lemma lt_succ_le_4 a b: T2nf b ->  (a < b) = (T2succ a <= b).
Lemma succ_nz x:  T2succ x != zero.
Lemma succ_psi a b: [a, b] != one -> T2succ [a,b] =  cons a b 0 one.
Lemma succ_psi_lt x a b : [a, b] != one ->  x < [a,b] -> T2succ x < [a,b].
Lemma succ_psi_lt2 a b x: [a, b] != one -> ([a, b] <= T2succ x) = ([a, b] <= x).
```

3.5 Addition

We define here addition and subtraction.

```
Fixpoint T2add x y :=
  if x is cons a b n c then
    if y is cons a' b' n' c' then
       if [a,b] < [a',b']  then y
       else if [a',b'] < [a,b] then cons a b n (c + y)
       else  cons a b (n+n').+1 c'
    else x
  else y
 where "x + y" := (T2add x y) : g0_scope.

Fixpoint T2sub x y :=
  if x is cons a b n c then
     if y is cons a' b' n' c' then
           if (x < y) then zero
           else if ([a',b'] < [a,b]) then x
           else if (n<n')%N then zero
           else if ([a,b]==one) then
             if (n==n')%N then zero else cons zero zero ((n-n').-1) zero
           else if(n==n') then c - c' else cons a b (n - n').-1 c
     else x
  else zero
where "a - b" := (T2sub a b) : g0_scope.
```

We show here some properties.

```
Lemma T2subn0 x: x - zero = x.
Lemma T2sub0n x: zero - x = zero.
Lemma T2subnn x: x - x = zero.
Lemma minus_le a b: a <= b -> a - b = zero.
Lemma nf_sub a b: T2nf a -> T2nf b -> T2nf (a - b).
Lemma sub_int n m : \F n - \F m = \F (n -m)%N.
Lemma succ_is_add_one a: T2succ a = a + one.
Lemma add1Nfin a:  ~~ T2finite a  -> one + a = a.
Lemma sub1Nfin a:  ~~ T2finite a  -> a - one  = a.
Lemma sub1a x: x != zero -> T2nf x -> x = one + (x - one).
Lemma sub1b x: T2nf x -> x = (one + x) - one.
Lemma T2add0n: left_id zero T2add.
Lemma T2addn0: right_id zero T2add.
```

```
Lemma add_int n m : \F n + \F m = \F (n +m)%N.
Lemma add_fin_omega n: \F n + omega = omega.
Lemma split_add x: let: (y,n) :=T2split x in T2nf x ->
   (x == y + \F n) && ((y==zero) || T2limit y ).
Lemma add_to_cons a b n c:
  c < [a,b] -> cons a b n zero + c = cons a b n c.
Lemma nf_add a b: T2nf a -> T2nf b -> T2nf (a + b).
Lemma T2add_eq0 m n:  (m + n == zero) = (m == zero) && (n == zero).
Lemma add_le1 a b: a <= a + b.
Lemma add_le2 a b: b <= a + b.
Lemma sub_le1 a b : T2nf a -> (a - b) <= a.
Lemma sub_pr a b: T2nf b -> (a + b) - a = b.
Lemma add_inj a b c : T2nf b -> T2nf c -> a + b = a + c -> b = c.
Lemma sub_pr1 a b: T2nf b -> a <= b -> b = a + (b - a).
Lemma omega_minus_one : omega - one = omega.
Lemma sub_nz a b: T2nf b -> a < b -> (b - a) != zero.
Lemma T2addA c1 c2 c3: c1 + (c2 + c3) = (c1 + c2) + c3.
Lemma T2le_add2l  p m n : (p + m <= p + n) = (m <= n).
Lemma T2lt_add2l  p m n : (p + m < p + n) = (m < n).
Lemma T2lt_add2r  p m n : (m + p  < n + p ) ->  (m < n).
Lemma T2le_add2r  p m n : (m <=n) -> (m + p  <= n + p).
Lemma T2eq_add2l  p m n : (p + m == p + n) = (m == n).
Lemma add_le3 a b: a = a + b -> b = zero.
Lemma add_le4 a b: b != zero -> a < a + b.
Lemma sub_pr1r a b: T2nf a -> a - b = zero -> a <= b.
```

3.6 The function ϕ

We say that x is AP if it has the form $[a,b]$; such numbers are additive principals (see previous chapter). The genera idea is that x is AP if and only if it is a power of ω [as mentioned above, we shall not define the power function in T_2].

```
Definition T2ap x :=
  if x is cons a b n c then ((n==0) && (c==zero)) else false.
Lemma ap_pr0 a b (x := [a,b]) u v:
  u < x -> v < x -> u + v < x.
Lemma ap_limit x: T2ap x -> (x == one) || (T2limit x).
Lemma ap_pr1 c:
   (forall a b,  a < c -> b < c -> a + b < c)  ->
   (c== zero) || T2ap c.
Lemma ap_pr2 c:
   T2nf c -> c <> zero ->
   (forall a b, T2nf a -> T2nf b ->  a < c -> b < c -> a + b < c)  ->
   T2ap c.
Lemma ap_pr3 a b y (x := [a,b]): y < x -> y + x = x.
Lemma ap_pr4 x: (forall b, b < x -> b + x = x) -> (x == zero) || T2ap x.
```

We define here a function of two arguments $\phi(a,b)$, sometimes written $\phi_a(b)$, related to $\psi(a,b)$, written $\psi_a(b)$ or $[a,b]$. Assume $b = \psi_u(v)+k$ where k is an integer, and $a < u$. If $k = 0$ we define $\phi_a(b) = b$, otherwise $\phi_a(b) = \psi_a(b-1)$. In all other cases, we define $\phi_a(b) = \psi_a(b)$.

```
Definition T2_pr1 x:= if x is cons a b n c then a else zero.
Definition T2_pr2 x:= if x is cons a b n c then b else zero.
```

```
Definition T2finite1 x:=
  if x is cons a b n c then [&& a == zero, b== zero & c == zero] else false.

Definition phi a b :=
   if b is cons u v n w then
     if ((n==0) && (w==zero)) then
        if (a < u) then b else [a,b]
     else if ((n==0) && (T2finite1 w) && (a <u))
       then [a, cons u v 0 (T2pred w) ]
     else [a,b]
   else [a,b].
```

We show here some trivial properties of ϕ. In particular, $\phi_a(b)$ has always the form $[u,v]$, where $a \leq u$, and $v \leq b$. If $a < u$, then $\phi_a(b) = b$. This says that b is a fix-point of ϕ_a. A consequence of $b < \psi(a,b)$ is: if b is a fix-point of ϕ_a, then $a < u$.

There are three cases: $\phi(a,b) = b$, or $\phi(a,b) = [a,b]$, or $\phi(a,b) = [a,b-1]$. The first case has been studied above. The last case if when $b = [u,v,0,k+1]$ where k is an integer, and $b-1 = [u,v,0,k]$.

```
Lemma phi_ap x y : (phi x y) = [T2_pr1 (phi x y), T2_pr2 (phi x y)].
Lemma phi_le1 a b: a <= T2_pr1 (phi a b).
Lemma phi_le2 a b: T2_pr2 (phi a b) <= b.
Lemma phi_le3 a b:  a < T2_pr1 (phi a b) -> (phi a b) = b.
Lemma phi_fix1 a u v: a < u -> phi a [u,v] = [u, v].
Lemma phi_fix2 a b (u:= T2_pr1 b) (v:= T2_pr2 b):
  phi a b = b -> b = [u,v] /\ a < u.
Lemma phi_succ a u v n:  a < u ->
  phi a (cons u v 0 (\F n.+1)) = [a, cons u v 0 (\F n)].
Lemma phi_cases a b:
    {phi a b = b} + {phi a b = [a, b]} +
    { phi a b = [a, T2pred b]  /\  b = T2succ (T2pred b)}.
Lemma nf_phi x y : T2nf x -> T2nf y -> T2nf (phi x y).
```

Let's show : every $[a,b]$ is in the image of ϕ_0, say is $\phi_0(c)$. This is obvious is a is non-zero (take $c = [a,b]$). Consider then $[0,b] = \phi_0(c)$. We can take $c = b$, except when $b = [a',b']$, where we take $[a',b']+1$, or when $b = [a',b'] + (k+1)$, where we take $[a',b'] + (k+2)$.

Fix a, and consider $P_a(x)$ the property that $\phi_c(x) = x$ whenever $c < a$. In the proof of what follows, we require some quantities to be NF; so we assume everything (a, c and x) to be NF. Obviously, $\phi(a,b)$ satisfies P_a. Conversely, assume that x satisfies P_a; let's exclude the case $a = 0$, so that $\phi_0(x) = x$, and x has the form $[a',b']$. If $c < a$; then $c < a'$. If a is a limit ordinal, this says $a \leq a'$. Assume a and x NF, so that a is the successor of a'' and $a'' \leq a'$. It is easy to see that $a'' = a'$ cannot be true, so that $a \leq a'$. The same argument as for ϕ_0 above shows that there is then b such that $x = \phi_a(b)$.

Thus, $P_a(x)$ is equivalent to: x is in the image of ϕ_a. As ϕ_a is strictly increasing (see below), ϕ_a is the enumeration of P_a. In the special case $a = 1$, we get: $\phi_0(x) = x$ if and only if x is in the range of ϕ_1.

```
Lemma phi_principalR a b: { c:T2 | [a, b] =  phi zero c}.
Theorem phi_spec1 a b c: c < a -> phi c (phi a b) = phi a b.
Lemma phi_spec2 a x:
    T2nf a -> T2nf x ->  (forall c,  T2nf c -> c < a -> phi c x = x) ->
    a <= T2_pr1 x.
```

```
Lemma phi_spec3 a x:
  T2nf a -> T2nf x -> (forall c,  T2nf c -> c < a -> phi c x = x) ->
  a != zero -> {b : T2 | x = phi a b}.
Lemma phi_spec4a u v: u != zero -> phi zero [u,v] = [u, v].
Lemma phi_spec4b x: phi zero x = x ->
  x = [T2_pr1 x, T2_pr2 x]  /\ T2_pr1 x != zero.
Lemma phi_spec4c x: phi zero x = x ->  { b: T2 | x = phi one b }.
```

We have $a < \phi_a(b)$. Note that Γ_0 is the least ordinal such that $a = \phi_a(0)$, so that all ordinals in T_2 are less than Γ_0. We also have $b \leq \phi_a(b)$. We have the non trivial property: ϕ_a is strictly increasing (we consider all possibilities, the non-trivial one is $\phi_a(b) = [a,b]$ and $\phi_a(b') = [a,b'-1]$. In this case b' has the form $[u,v,0,k]$ where k is an integer. must show $b < b'-1$, knowing $b < b'$; in other words, we must show that $b' = b+1$ is absurd; this works only if b is NF. One deduces a criterion (assuming everything NF) for $\phi_a(b) = \phi_c(d)$ or $\phi_a(b) < \phi_c(d)$. The last criterion is (3.1).

```
Lemma no_critical a: a < phi a zero.
Lemma phi_ab_le1 a b: b <= phi a b.
Lemma phi_ab_le2  a b: a < phi a b.
Lemma phi_inv1 a b: phi a (T2succ b) = [a,b] ->
   { n: nat | (b =  cons (T2_pr1 b) (T2_pr2 b) 0 (\F n) /\ a < T2_pr1 b) }.

Lemma phi_mono_a a b b': T2nf b -> b < b' ->  phi a b < phi a b'.
Lemma phi_mono_b a b b': T2nf b -> b <= b' -> phi a b <= phi a b'.

Lemma phi_mono_c a b b': T2nf b -> T2nf b' ->
  (phi a b < phi a b') = (b < b').
Lemma phi_inj a b b': T2nf b -> T2nf b' -> phi a b = phi a b' -> b = b'.
Lemma phi_inj1 a b b': T2nf b -> T2nf b' -> (phi a b == phi a b') = (b== b').

Lemma phi_eqE a b a' b': T2nf a -> T2nf a' ->  T2nf b -> T2nf b' ->
   (phi a b == phi a' b') =
    (if a < a' then  b == phi a' b'
     else if a' < a then phi a b == b' else  b== b').
Lemma phi_ltE a b a' b': T2nf a -> T2nf a' ->  T2nf b -> T2nf b' ->
   (phi a b < phi a' b') =
    (if a < a' then  b < phi a' b'
     else if a' < a then phi a b < b' else  b < b').
```

Let's show: every x of the form $[y,z]$ has the form $\phi_a(b)$ where $b < x$; this form is unique and a and b are structurally smaller than x.

First, if $x = \phi(a,b)$ and $b < x$, then $x = [a,b']$, where b' is b or $b-1$. In particular, $\phi(a,b) = \phi(a',b')$ says $a = a'$; it follows $b = b'$.

```
Lemma phi_inv0  a b a' b':
  phi a b = phi a' b' -> b < phi a b -> b' < phi a' b' -> a = a'.
Lemma phi_inv2 a b a' b':
  phi a b = phi a' b' -> b < phi a b -> b' < phi a' b' -> b = b'.
Lemma phi_inv3 x:
 T2ap x -> { a: T2 &  { b: T2 |
    [/\ x = phi a b, b < x,  (size a <  size x)%N & (size b < size x)%N ] }}.
```

Let a and b be two ordinals. If $b = b_0 + n$, where n is an integer and $\acute{L}phi(a,b) = b_0$, we set $b' = b+1$ else $b' = b$. We define $\psi'_a(b)$ to be $\phi(a,b')$. We want to show $\psi = \psi'$. We first notice $b' < \phi(a,b')$.

```
Definition psi_phi_aux a b :=
  let (b', n) := T2split b in if phi a b' == b' then (T2succ b) else b.
Definition psi_phi a b := phi a (psi_phi_aux a b).

Lemma psi_phi1 a b (c:= psi_phi_aux a b): c < phi a c.
```

Note: this needs to be completed.

Chapter 4

The third model

4.1 Introduction

We implement here the ordinals defined by [1]. He consider the least set A satisfying: 1 is in A; if α and β are in A, so is $\alpha+\beta$, and if moreover γ is in A, so is (α,β,γ). If $2 = 1+1$, then $1+2$ and $2+1$ are in A. These two objects are considered equal, in other terms, addition is *associative*. This can be restated as: every element of A has the form

$$\pi_1 + \pi_2 + \ldots + \pi_m, \tag{4.1}$$

where this means π_1 if $m = 1$, and each π_k is an element of A of the form (α,β,γ), or is 1. We shall define below an ordering, and consider only the elements of A for which the π_k are in decreasing order. This set will be denoted by B.

Let's write π^k as short of $\pi+\pi+\ldots+\pi$ (there are $k+1$ terms in the sum), where $\pi^0 = \pi$. Now, every element of A has the form

$$x = \pi_1^{n_1} + \pi_2^{n_2} + \ldots + \pi_m^{n_m}, \tag{4.2}$$

where π_k is as above, and different from π_{k-1} and π_{k+1}. An element of B has the same form, where the sequence π_k is strictly decreasing. Assume $\pi_1 = (\alpha,\beta,\gamma)$. Let δ be $\pi_2^{n_2} + \ldots + \pi_m^{n_m}$. We denote by $[\alpha,\beta,\gamma,n_1,\delta]$ the object x. In the special case where $m = 1$, we shall denote x by $[\alpha,\beta,\gamma,n_1,0]$. In the special case where $\pi_1 = 1$, we shall denote x by $[0,0,0,n_1,\delta]$.

Now, every element of A can be uniquely written in the form $[\alpha,\beta,\gamma,n,\delta]$, where $n > 0$, other quantities are in A or zero. Let $A^+ = A \cup \{0\}$, with the rules $x+0 = 0+x = x$, and $0 < x$ for $x \in A$ (the relation $<$ will be defined below). So every non-zero element of A^+ can be written as $[\alpha,\beta,\gamma,n,\delta]$, where α, β, γ, and δ are in A^+. The converse is false (if one of α, β, γ, is zero, but not all of them, there is a problem).

We shall study here the set A' formed of 0 and all $[\alpha,\beta,\gamma,n,\delta]$, where α, β, γ, and δ are in A'. We shall deduce a set B', then try to find an interpretation for the expression (α,β,γ). For instance $(1,1,\gamma) = \omega^\gamma$.

We start be defining our induction type and equality.

```
Inductive T3 : Set :=
  zero : T3
| cons : T3 -> T3 -> T3 -> nat -> T3 -> T3.

Fixpoint T3eq x y {struct x} :=
```

```
  match x, y with
  | zero, zero  => true
  | cons a b c n d, cons a' b' c' n' d' =>
      [&& T3eq a a', T3eq b b', T3eq c c', n== n' & T3eq d d' ]
  | _, _ => false
end.

Definition T3nat p := if p is n.+1 then cons zero zero zero n zero else zero.

Notation "\F n" := (T3nat n)(at level 29) : ak_scope.
Notation "[ x , y , z ]" := (cons x y z 0 zero) (at level 0) :ak_scope.

Lemma T3eqP : Equality.axiom T3eq.
Lemma T3eqE a b c n d a' b' c' n' d':
  (cons a b c n d == cons a' b' c' n' d') =
      [&& a == a', b == b', c == c', n== n' & d == d' ].
```

We define $l(x)$ to be the height of x, as in the previous case, and give two properties.

```
Lemma size_a a b c n d: size a < size (cons a b c n d).
Lemma size_b a b c n d: size b < size (cons a b c n d).
Lemma size_c a b c n d: size c < size (cons a b c n d).
Lemma size_d a b c n d: size d < size (cons a b c n d).
Lemma size_psi a b c n d: size [a, b, c] <= size (cons a b c n d).
Lemma size_prop1 a b c n d (l:= size (cons a b c n d)):
   [&& size a < l, size b < l, size c < l,  size d < l
   & size [a, b, c] <= l]%N.
Lemma size_prop a b c n d a' b' c' n' d'
   (l := size (cons a b c n d) + size (cons a' b' c' n' d')) :
  [&& (size a' + size a < l), (size b + size b' < l),
   (size c + size c' < l), (size d + size d' < l),
   (size a + size a' < l), (size b' + size b < l),
   (size [a, b, c] + size b' < l),(size b + size [a', b', c'] < l),
   (size [a, b, c] + size c' < l) &(size c + size [a', b', c'] < l)].
```

4.2 Comparison

Ackerman has six rules, G1, G2, G3, G4, G5, and G6; the last rule has 9 sub-rules, from G6a to G6i. These rules show how to compute the quantity $x < x'$, denoted by c. It is obvious that c is false if $x = x'$. Assume x and x' different; then $x' < x$ is the negation of c.

The first two rules say that $x < 1$ is false, and $1 < x$ holds iff $x \neq 1$ (remember that Ackermann considers only non-zero ordinals; so for us $x < 0$ is false, and $0 < x$ holds iff $x \neq 0$). Rules G3, G4 and G5 consider the case of two sequences of the form (4.1). If the elements are π_i and π'_i, with size m and m' then, if all the terms are the same, then c is $m < m'$. If i is the least index which π_i and π'_i differ, then c is $\pi_i < \pi'_i$. We can restate this as follows. Assume $x = [\alpha, \beta, \gamma, n, \delta]$, and let $y = (\alpha, \beta, \gamma)$. If $y \neq y'$, then c is $y < y'$, otherwise, if $n \neq n'$, then c is $n < n'$, otherwise it is $\delta < \delta'$.

It suffices to give a rule for $y < y'$. Rules G1 and G2 consider the case where one argument is 1. Rule G6 consider the case $x = (\alpha, \beta, \gamma)$ and $x' = (\alpha', \beta', \gamma')$. It says that $x < x'$ is equivalent to one of the following

a) $\alpha = \alpha', \beta = \beta'$ and $\gamma < \gamma'$

b) $\alpha = \alpha', \beta < \beta'$ and $\gamma < x'$

c) $\alpha = \alpha', \beta' < \beta$ and $x < \gamma'$

d) $\alpha = \alpha', \beta' < \beta$ and $x = \gamma'$

e) $\alpha < \alpha', \beta < x', \gamma < x'$

f) $\alpha' < \alpha, x < \beta'$

g) $\alpha' < \alpha, x = \beta'$

h) $\alpha' < \alpha, x < \gamma'$

i) $\alpha' < \alpha, x = \gamma'$

These formulas contain x and x', so that they cannot be defined by induction on x nor on y. We apply the same method as for the second model. Since the formulas are more complicated, we first write G6, using f instead of $<$, then the specification of f.

```
Definition lt_psi_rec f a b c a' b' c' (x := [a,b,c])(x':= [a', b', c']):=
  [|| [&& a==a', b==b' & f c c'],
      [&& a==a', f b b' & f c x'],
      [&& a==a', f b' b & f x c'],
      [&& a==a', f b' b & x == c'],
      [&& f a a', f b x' & f c x'],
      ((f a' a) && f x b'),
      ((f a' a) && (x == b')),
      ((f a' a) && f x c') |
      ((f a' a) && (x == c'))].

Definition lt_rec f x y :=
 if x is cons a b c n d then
   if y is cons a' b' c' n' d' then
     if (lt_psi_rec f a b c a' b' c')
     then true
     else if ((a== a') && (b==b') && (c==c')) then
       if (n < n')%N then true
       else if (n == n') then  (f d d') else false
       else false
   else false
 else if y is cons a' b' c' n' d' then true else false.
```

The comparison is now the following.

```
Fixpoint T3lta k {struct k}:=
 if k is k.+1 then lt_rec (T3lta k) else fun x y => false.

Definition T3lt a b :=  T3lta ((size a) + size b).+1 a b.
Definition T3le (x y :T3) := (x == y) || (T3lt x y).
Notation "x < y" := (T3lt x y) : ak_scope.
Notation "x <= y" := (T3le x y) : ak_scope.
Notation "x >= y" := (y <= x) (only parsing) : ak_scope.
Notation "x > y"  := (y < x) (only parsing)  : ak_scope.
```

We rewrite now condition G6 using <. We obtain a simpler formula by merging G6c and G6d, G6f and G6g, G6h and G6i, and using ≤. The last lemma says: let $x = [a,b,c,n,d]$ and $y = [a,b,c]$. Let x' and y' be defined similarly. We have then $x < x'$ (resp: $x \le x'$) if either $y < y'$, or $y = y'$ and then, either $n < n'$ or $n = n'$ and $d < d'$ (resp: $d \le d'$). Note: the two lemmas `T3lt_psi'` and `T3lt_consE` depend on `T3ltgtP`; we moved them here for convenience.

```
Definition lt_psi (a b c a' b' c': T3):=
 [|| [&& a==a', b==b' & c < c'],
      [&& a==a', b < b' & c < [a',b',c']],
      [&& a==a', b' < b & [a,b,c] < c'],
      [&& a==a', b' < b & [a,b,c] == c'],
      [&& a < a', b < [a',b',c'] & c < [a',b',c']],
      ((a' < a) && ([a,b,c] < b')),
      ((a' < a) && ([a,b,c] == b')),
      ((a' < a) && ([a,b,c] < c')) |
      ((a' < a) && ([a,b,c] == c'))].
Lemma T3ltE x y : x < y  = lt_rec T3lt x y.
Lemma T3lt_psi a b c a' b' c': [a,b,c] < [a', b',c'] = lt_psi a b c a' b' c'.

Lemma T3lt_psi' a b c a' b' c': [a, b, c] < [a', b', c' ] =
  [|| [&& a==a', b==b' & c < c'],
      [&& a==a', b < b' & c < [a', b', c'] ],
      [&& a==a', b' <b & [a,b,c] <= c'],
      [&& a < a', b < [a', b', c'] & c < [a', b', c']],
      ((a' < a) && ([a,b,c] <= b')) |
      ((a' < a) && ([a,b,c] <= c'))].
Lemma T3lt_consE a b c n d  a' b' c' n' d' :
   cons a b c n d < cons a' b' c' n' d' =
     if ([a, b, c] < [a', b', c']) then true
     else if ([a, b, c] == [a', b', c']) then
       if (n < n')%N then true
       else if (n == n') then  (d < d') else false
       else false.
Lemma T3le_consE a b c n d  a' b' c' n' d' :
   cons a b c n d <= cons a' b' c' n' d' =
     if ([a, b, c] < [a', b', c']) then true
     else if ([a, b, c] == [a', b', c']) then
       if (n < n')%N then true
       else if (n == n') then  (d <= d') else false
       else false.
```

Let's define $\omega = [0,0,1]$, $\epsilon_0 = [0,1,0]$ and $b = [0,0,0,0,1]$. The quantity b is an example of non NF ordinal.

```
Definition one := [zero,zero,zero].
Definition omega := [zero,zero, one].
Definition epsilon0 :=  [zero, one, zero].
Definition T3bad := cons zero zero zero 0 one.
```

Some trivial lemmas.

```
Lemma T3ltn0 x: (x < zero) = false.
Lemma T3lt0n x: (zero < x) = (x != zero).
Lemma T3le0n x: zero <= x.
Lemma T3len0 x: (x <= zero) = (x == zero).
```

```
Lemma T3ltnn x: (x < x) = false.
Lemma T3lt_ne a b : a < b -> (a == b) = false.
Lemma T3lt_ne' a b : a < b -> (b == a) = false.
Lemma T3ltW a b : (a < b) -> (a <= b).
Lemma T3le_eqVlt a b : (a <= b) = (a == b) || (a < b).
Lemma T3lt_neAle a b : (a < b) = (a != b) && (a <= b).
Lemma T3ge1 x:  (one <= x) = (x != zero).
Lemma T3lt1 x: (x < one) = (x==zero).
Lemma T3lcp0_pr x y: x < y -> (y==zero) = false.
Lemma finite_ltP n p : (n < p)%N = (\F n < \F p).
```

Claim I of [1] is that $x < x$ is always false (obvious by structural induction). We show here claims II and III (by induction on the sum of the heights of the arguments).

```
Lemma T3lt_anti b a: a < b -> (b < a) = false.
Lemma T3lt_trichotomy a b: [|| (a< b), (a==b) | (b < a)].

Lemma T3lenn x: x <= x.
Lemma T3leNgt a b:  (a <= b) = ~~ (b < a).
Lemma T3ltNge a b:  (a < b) = ~~ (b <= a).
Lemma T3eq_le m n : (m == n) = ((m <= n) && (n <= m)).
Lemma T3leP x y : T3leq_xor_gtn x y (x <= y) (y < x).
Lemma T3ltP m n : T3ltn_xor_geq m n (n <= m) (m < n).
Lemma T3ltgtP m n : compare_T3 m n (m < n) (n < m) (m == n).
```

Claim IV is transitivity of comparison. Proof by induction on the sum of the sizes of the arguments and case analysis. (long, 200 lines)

```
Theorem T3lt_trans b a c: a < b -> b < c -> a < c.

Lemma T3lt_le_trans b a c: a < b -> b <= c -> a < c.
Lemma T3le_lt_trans b a c: a <= b -> b < c -> a < c.
Lemma T3le_trans b a c: a <= b -> b <= c -> a <= c.
Lemma T3le_anti : antisymmetric T3le.
Lemma T3le_total m n : (m <= n) || (n <= m).
```

Claims V and VI are equivalent to: each of a, b and c is less than $[a,b,c]$. Proof. We first show that if $x \leq b$ or $x \leq c$, then $x < [a,b,c]$, by induction on the size of x. Consider the case $x \leq b$. Assume $x = [a',b',c',n',d']$. It suffices to show $[a',b',c'] < [a,b,c]$. Let $y = [a',b',c']$. By induction $b' < y$ and $c' < y$. By assumption $y < b$, so that $b' < b$ and $c' < b$. By induction, b' and c' are less than $[a,b,c]$. The result follows, by comparing a and a'. The case $x \leq c$ is similar (if $a = a'$, we have to compare b and b'). Assume now $x \leq a$. The same argument as above gives $a' < a$. By induction and claim V, b' and c' are less than $[a,b,c]$. The result follows from G6e.

```
Lemma T3le_psi a b c n d: [a,b,c] <= cons a b c n d.
Lemma T3lt_psi_bc  a b c: ((b < [a,b,c]) && (c < [a, b, c])).
Lemma T3lt_psi_b a b c: b < [a,b,c].
Lemma T3lt_psi_c a b c: c < [a,b,c].
Lemma T3lt_psi_a a b c: a < [a,b,c].
```

4.3 Accessibility

Ackermann defines "O-Zahl" (we shall write here NF) by induction as: 1 is NF, $\pi_1+\pi_2+\ldots+\pi_i$ is NF if each π_k is NF, and the sequence is decreasing, and (α,β,γ) is NF if and only if each α, β, γ is NF.

With our notations, $x=[a,b,c,n,[a',b',c',n',d']]$ corresponds to $\pi_1^n+\pi_2^{n'}+\ldots$, where π^n is the sum of $n+1$ occurrences of π. We shall assume $\pi_1\neq\pi_2$, so that (by induction of the number of terms in the sum), x is NF if and only if π_1 is NF, $\pi_2^{n'}+\ldots$ is NF and $\pi_1<\pi_2$. Thus, we define $[a,b,c,n,d]$ to be NF if a, b, c, d are NF and $d<[a,b,c]$.

```
Fixpoint T3nf x :=
  if x is cons a b c _ d
  then [&& T3nf a, T3nf b, T3nf c, T3nf d & d < [a,b,c] ]
  else true.

Lemma nf_0: T3nf zero.
Lemma nf_int n: T3nf (\F n).
Lemma nf_psi a b c: T3nf [a, b, c] = [&& T3nf a, T3nf b & T3nf c].
Lemma nf_consE a b c n d:
    T3nf(cons a b c n d) = [&& T3nf [a,b,c], T3nf d & d < [a,b,c] ].

Lemma nf_cons_cons a b c n a' b' c' n' d':
  T3nf(cons a b c n (cons a' b' c' n' d')) =
   [&& [a', b',c'] < [a, b,c], T3nf [a,b, c] &
    T3nf (cons a' b' c' n' d') ].
```

Ackermann claims "Alle O-Zahlen sind erreichbar" (all NF-ordinals are accessible), using 13 lemmas, TI through TXIII. Let's write $\mathrm{N}(x)$ for : x is NF, $\mathrm{A}(x)$ for: x is accessible, and $x<_{\mathrm{N}}y$ for: x and y are NF and $x<y$. TII says: if $\mathrm{A}(x)$ and $y<_{\mathrm{N}}x$ then $\mathrm{A}(y)$, TIII says: if, for every y such that $y<_{\mathrm{N}}x$, we have $\mathrm{A}(y)$, then $\mathrm{A}(x)$ holds. These two properties says that $\mathrm{A}(x)$ is (Acc $<_{\mathrm{N}}$ x), where Acc is the Coq accessibility property, and the claim is just that $<_{\mathrm{N}}$ is well-founded.

Lemma TI says A(0), which is trivial. Thus, in order to prove $\mathrm{A}(x)$, it suffices to show $\mathrm{A}(x')$, whenever $x'<_{\mathrm{N}}x$ and x' is non-zero.

Lemma TV says: let $x=[a,b,c,n,d]$, and $y=[a,b,c]$; assume $\mathrm{A}(y)$ and $\mathrm{N}(x)$; then $\mathrm{A}(x)$. Let $y_n=[a,b,c,n,0]$. Since $x<_{\mathrm{N}}y_{n+1}$, it suffices to show $\mathrm{A}(y_{n+1})$. Thus, we show $\mathrm{A}(y_n)$ by induction on n. Let $x'=[a',b',c',n',d']$ and $y'=[a',b',c']$; assume $\mathrm{N}(x')$. The key relation is that, if $x'<x$, then $y'\leq y$. If $y'<y$ then $x'<y$, thus $\mathrm{A}(x')$. If $y'=y$ and $n'<n$ then $x'<y_n$, thus $\mathrm{A}(x')$. We are left with $x'=[a,b,c,n,d']$. From $\mathrm{N}(x')$ we deduce $d'<_{\mathrm{N}}y$ thus $\mathrm{A}(d')$. We now proceed by induction on $\mathrm{A}(d')$. We get as assumption, that, for every d'' such that $d''<_{\mathrm{N}}d_1$, we have $\mathrm{A}(d'')$ and $\mathrm{A}([a,b,c,n,d''])$. We must show $\mathrm{A}([a,b,c,n,d_1])$. Let $x_1=[a,b,c,n,d_1]$ and $x''=[a'',b'',c'',n'',d'']$, $y''=[a'',b'',c'']$. We must show that $x''<_{\mathrm{N}}x_1$ implies $\mathrm{A}(x_1)$. The case $y''<y$ is as above; the case $n''<n$ is above. The last case is trivial.

Lemma TIX says: assume $[a,b,c]$ NF, $\mathrm{A}(b)$, $\mathrm{A}(c)$. Assume $\mathrm{A}([a',b',c'])$ whenever either $a'<_{\mathrm{N}}a$, or $a'=a$ and $b'<_{\mathrm{N}}b$ or $a'=a$, $b'=b$ and $c'<_{\mathrm{N}}c$ (assuming $\mathrm{A}(b')$ in the first two cases, $\mathrm{A}(c')$ in the first case). Then $\mathrm{A}([a,b,c])$ holds. Proof. We must show $\mathrm{N}(x')$, assuming $x'<[a,b,c]$. We proceed by induction on the size of x'. We may assume (by TV) that $x'=[a',b',c']$. There are different reasons why $[a',b',c']<[a,b,c]$ holds. For instance, we may have $a<a'$ and $x'<b$. Here we conclude by $\mathrm{A}(b)$. In the case $a'=a$, $b'<b$ and $c'<x$, we apply our assumption. It requires $\mathrm{N}(c')$, which follows by $c'<[a,b,c]$ and our assumption on

the size of x'.

By induction on A(c), the third condition ($a' = a$, $b' = b$ and $c' <_N c$) is un-necessary. By induction on A(b), the second condition is un-necessary. By induction on A(a), the first condition is un-necessary. This gives lemma TXII.

We terminate by induction on the size k of x. We must show A(y), whenever $y < x$. By TV, we may assume $y = [a,b,c]$. By TXII, if suffices to show A(a), A(b) and A(c). But this holds by induction on k.

```
Lemma nf_Wf : well_founded_P T3nf T3lt.
```

Consider the function $f : T_1 \to T_3$ defined by $f(0) = 0$ and $f([a,n,b]) = [0,0,f(a),n,f(b)]$. It is strictly increasing, thus injective; the values are less than ϵ_0. Since $<$ is not well-founded in T_1, it is neither well-founded in T_3.

Note that $x < \epsilon_0$ if x is zero or $[0,0,c,n,d]$ with $c < \epsilon_0$. If x is NF, we have also $d < \epsilon_0$. Moreover, any NF x less than ϵ_0 has the form $f(x)$. Thus the set of ordinals $< \epsilon_0$ is isomorphic to T_1.

```
Fixpoint T1_T3 (c:CantorOrdinal.T1) : T3 :=
  if c is CantorOrdinal.cons a n b then cons zero zero (T1_T3 a) n (T1_T3 b)
  else zero.

Definition all_zero a b c :=[&& a==zero, b==zero & c== zero].
Lemma all_zeroE a b c: all_zero a b c = ([a,b,c] == one).
Theorem lt_not_wf : ~ (well_founded T3lt).
Lemma T1T3_inc x y: (x <y)%ca-> (T1_T3 x) < (T1_T3 y).
Lemma T1T3_lt_epsilon0 x: T1_T3 x < epsilon0.
Lemma TT1T3_inj: injective T1_T3.
Lemma T1T3_surj x: T3nf x -> x < epsilon0 -> exists y, x = T1_T3 y.
```

Let's define limit, finite and how to split x as $l(x)$ and $c(x)$. If x is zero, it is finite, not limit, l and c are zero. If $x = [a,b,c,n,d]$, we meet the condition $a = b = c = 0$; it can be replaced by $[a,b,c] = 1$. If the condition holds, and x is NF, then $d = 0$, thus $x = F(n+1)$. If the condition holds, then x is finite, not limit, $l(x) = 0$, and $c(x) = n+1$. If the condition is false, then x is not finite, it is limit if either $d = 0$ or d is limit. We have $l(x) = [a,b,c,n,l(d)]$ and $c(x) = c(d)$.

```
Definition all_zero a b c :=(a==zero) && (b==zero) && (c== zero).

Fixpoint T3limit x :=
  if x is cons a b c n d then
    if (all_zero a b c) then false else (d== zero) || T3limit d
  else false.

Definition T3finite x :=
   if x is cons a b c n d then all_zero a b c else true.

Fixpoint T3split x:=
 if x is cons a b c n d then
      if  all_zero a b c then (zero, n.+1) else
     let: (x, y) := T3split d in (cons a b c n x,y)
   else (zero,0).
```

Denote by ω_x the quantity $[0,0,x]$ Then $\omega_x < [a,b,c]$ if either $a = b = 0$ and $c < c'$, or, when one of a, is non-zero, then $x < [a,b,c]$.

```
Lemma split_limit x: ((T3split x).2 == 0) = ((x==zero) || T3limit x).
Lemma split_finite x: ((T3split x).1 == zero) = T3finite x.
Lemma T3nf_finite a b c n d:  all_zero a b c -> T3nf (cons a b c n d) ->
    d = zero.
Lemma T3finite1 n: T3finite (\F n).
Lemma T3finite2 x: T3finite x -> T3nf x -> x = \F ((T3split x).2).


Lemma T3gt1 x: (one < x) = ((x != zero) && (x != one)).
Lemma omega_least_inf1 x: T3finite x -> x < omega.
Lemma omega_least_inf2 x: ~~ T3finite x -> omega <= x.

Lemma lt_omega1 c n d a' b' c' n' d' :
   cons zero zero c n d < cons a' b' c' n' d' =
     if ((a'== zero) && (b'==zero)) then
       ((c < c') || ((c==c') && ((n < n')%N || ((n==n') && (d < d')))))
    else (c < [a', b', c']).
Lemma lt_omega2 c a' b' c' :
   ([zero, zero, c] < [a', b', c']) =
     if ((a'== zero) && (b'==zero)) then c < c' else  (c < [a', b', c']).
Lemma lt_epsilon0 a b c n d :
  cons a b c n d < epsilon0 = [&& a==zero, b == zero & c < epsilon0 ].
```

Let's define the successor and predecessor as in the case T_1.

```
Fixpoint T3is_succ x :=
  if x is cons a b c n d then (all_zero a b c) || T3is_succ d  else false.

Fixpoint T3succ x :=
  if x is cons a b c n d
     then if all_zero a b c then \F n.+2  else cons a b c n (T3succ d)
  else one.

Fixpoint T3pred x :=
  if x is cons a b c n d then
     if  all_zero a b c  then \F n else (cons a b c n (T3pred d))
  else zero.
```

We have the same properties as in the case T_1. In particular, there is no NF y such that $x < y < x^+$.

```
Lemma split_is_succ x: ((T3split x).2 != 0) = (T3is_succ x).
Lemma split_succ x: let:(y,n):= T3split x in T3split (T3succ x) = (y,n.+1).
Lemma split_pred x: let:(y,n):= T3split x in T3split (T3pred x) = (y,n.-1).
Lemma split_le x :  (T3split x).1 <= x.
Lemma nf_split x : T3nf x -> T3nf (T3split x).1.
Lemma T3finite_succ x: T3finite x -> T3finite (T3succ x).
Lemma T1succ_nat n: T3succ (\F n) = \F (n.+1).
Lemma nf_omega : T3nf omega.
Lemma nf_finite n: T3nf (\F n).
Lemma limit_pr1 x: (x == zero) (+) (T3limit x (+) T3is_succ x).
Lemma limit_pr x y: T3limit x -> y < x -> T3succ y < x.
Lemma pred_le a: T3pred a <= a.
Lemma pred_lt a: T3is_succ a -> T3pred a < a.
Lemma succ_lt a: a < T3succ a.
Lemma succ_nz x:  T3succ x != zero.
```

```
Lemma all_zeroE a b c: all_zero a b c = ([a,b,c] == one).
Lemma nf_succ a: T3nf a -> T3nf (T3succ a).
Lemma nf_pred a: T3nf a -> T3nf (T3pred a).
Lemma succ_pred x: T3nf x -> T3is_succ x -> x = T3succ (T3pred x).
Lemma succ_p1 x: T3is_succ (T3succ x).
Lemma pred_succ x: T3nf x -> T3pred (T3succ x) = x.
Lemma succ_inj x y: T3nf x -> T3nf y -> (T3succ x == T3succ y) = (x==y).
Lemma lt_succ_succ x y: T3succ x < T3succ y -> x < y.
Lemma le_succ_succ x y: x <= y -> T3succ x <= T3succ y.
Lemma lt_succ_succE x y:
  T3nf x -> T3nf y ->  (T3succ x < T3succ y) = (x < y).
Lemma le_succ_succE x y:
  T3nf x -> T3nf y -> (T3succ x <= T3succ y) = (x <= y).
Lemma lt_succ_le_1 a b : T3succ a <= b ->  a < b.
Lemma lt_succ_le_2 a b:  T3nf a -> a < T3succ b ->  a <= b.
Lemma lt_succ_le_3 a b:  T3nf a -> (a < T3succ b) = (a <= b).
Lemma lt_succ_le_4 a b: T3nf b ->  (a < b) = (T3succ a <= b).

Lemma succ_prop x y: T3nf y -> x < y -> y < T3succ x -> false.
Lemma succ_psi a b c: [a, b, c] != one -> T3succ [a,b,c] =  cons a b c 0 one.
Lemma succ_psi_lt x a b c: [a, b, c] != one ->
   x < [a,b,c] -> T3succ x < [a,b,c].
Lemma succ_psi_lt2 a b c x: ([a, b, c] != one ->
  ([a, b, c] <= T3succ x) = ([a, b, c] <= x).
```

4.4 Addition

The definition of addition is obvious, that of subtraction is a bit less.

```
Fixpoint T3add x y :=
  if x is cons a b c n d then
    if y is cons a' b' c' n' d' then
       if [a,b,c] < [a',b',c']  then y
       else if [a',b',c'] < [a,b,c] then cons a b c n (d + y)
       else  cons a b c (n+n').+1 d'
    else x
  else y
 where "x + y" := (T3add x y) : ak_scope.

Fixpoint T3sub x y :=
  if x is cons a b c n d then
     if y is cons a' b' c' n' d' then
           if (x < y) then zero
           else if ([a',b',c'] < [a,b,c]) then x
           else if (n<n')%N then zero
           else if ([a,b,c] == one) then
             if (n==n')%N then zero else cons zero zero zero ((n-n').-1) zero
           else if(n==n') then d - d' else cons a b c (n - n').-1 d
     else x
  else zero
where "a - b" := (T3sub a b) : ak_scope.
```

By induction $x - x = 0$, thus $x - y$ is zero when $x \leq y$.

```
Lemma T3subn0 x: x - zero = x.
```

```
Lemma T3sub0n x: zero - x = zero.
Lemma T3subnn x: x - x = zero.
Lemma minus_lt a b: a < b -> a - b = zero.
Lemma minus_le a b: a <= b -> a - b = zero.
Lemma nf_sub a b: T3nf a -> T3nf b -> T3nf (a - b).
Lemma sub_int n m : \F n - \F m = \F (n -m)%N.
```

Some properties of addition.

```
Lemma succ_is_add_one a: T3succ a = a + one.
Lemma add1Nfin a:  ~~ T3finite a  -> one + a = a.
Lemma sub1Nfin a:  ~~ T3finite a  -> a - one = a.
Lemma sub1a x: x != zero -> T3nf x -> x = one + (x - one).
Lemma sub1b x: T3nf x -> x = (one + x) - one.
Lemma T3add0n : left_id zero T3add.
Lemma T3addn0: right_id zero T3add.
Lemma add_int n m : \F n + \F m = \F (n +m)%N.

Lemma sub_1aCE (a:= T3bad) : one + (a - one) != a.
Lemma sub_1bCE (a:= T3bad) : (one + a - one) != a.
```

We have $n+\omega = \omega$ if n is finite. In particular, addition is non-commutative. If we split x as $l(x)$ and $c(x)$ then $x = l(x)+c(x)$, and $l(x)$ is zero or limit. On the other hadd, if $x = [a,b,c,n,d]$ and $y = [a,b,c,n,d]$ then $x = y+d$.

```
Lemma add_fin_omega n: \F n + omega = omega.
Lemma split_add x: let: (y,n) :=T3split x in T3nf x ->
   (x == y + \F n) && ((y==zero) || T3limit y ).
Lemma add_to_cons a b c n d:
  d < [ a,b,c] -> cons a b c n zero + d = cons a b c n d.
Lemma addC_CE (a := one) (b := omega):
  [&& T3nf a, T3nf b & a + b != b + a].

Lemma fooCE (x:= T3bad):
   ~~T3limit x  /\ (forall u v, T3limit u -> x <> u + \F v.+1).

Lemma nf_add a b: T3nf a -> T3nf b -> T3nf (a + b).
Lemma T3add_eq0 m n: (m + n == zero) = (m == zero) && (n == zero).
Lemma add_le1 a b: a <= a + b.
Lemma add_le2 a b: b <= a + b.
Lemma sub_le1 a b : T3nf a -> (a - b) <= a.
Lemma sub_pr a b: T3nf b -> (a + b) - a = b.
Lemma add_inj a b c : T3nf b -> T3nf c -> a + b = a + c -> b = c.
Lemma sub_pr1 a b: T3nf b -> a <= b -> b = a + (b - a).
Lemma sub_pr1r a b: T3nf a -> a - b = zero -> a <= b.
Lemma sub_nz a b: T3nf b -> a < b -> (b - a) != zero.
Lemma omega_minus_one : omega - one = omega.

Lemma T3addA c1 c2 c3: c1 + (c2 + c3) = (c1 + c2) + c3.
Lemma T1addS a b : (a + T1succ b) = T1succ (a+ b).
Lemma T3le_add2l  p m n : (p + m <= p + n) = (m <= n).
Lemma T3lt_add2l  p m n : (p + m < p + n) = (m < n).
Lemma T3lt_add2r  p m n : (m + p  < n + p ) ->  (m < n).
Lemma add_le3 a b: a = a + b -> b = zero.
Lemma add_le4 a b: b != zero -> a < a + b.
```

4.5 Limit ordinals

As mentioned above, every countable limit ordinal x is the supremum of a strictly increasing sequence of ordinals x_k. By the axiom of choice, there is a function $f(x,n)$ such that $x_k = f(x,n)$. The function is obviously non-unique (every infinite subsequence of x_k has the same limit as x_k). It happens that we can construct a function f, valid on T_3. The quantity $f(x,n)$ is denoted $\phi_n(x)$ by Ackermann.

Consider the condition

$$\mathrm{H}_b(a,b,c) \iff c = [a_1,b_1,c_1] \text{ and } (a = a_1 \wedge b < b_1) \text{ or } (a < a_1 \wedge b < c) \tag{4.3}$$

For simplicity, ϕ_n stands for $\phi_n(x)$. We have replaced one by zero, except in the case $x+1$; and we have replaced $(0,0,0)$ by ω. This conflicts with line 3, $\gamma = 0$, see below. In the table, α, β and γ are any ordinals, π has the form $[\alpha,\beta,\gamma]$, λ and μ are limit ordinals. The case 12 is characterized by $x = [a,b,c]$, where c is limit ordinal; If $\mathrm{H}_b(a,b,c)$ is false we are in case a, otherwise in case b, and there are five subcases. In the cases 1, 5 and 12a, the last column is missing; in these cases the third column gives the value of $\phi_n(x)$ for every n.

	x	$\phi_0(x)$	$\phi_{n+1}(x)$
1	$\alpha+\pi$	$\alpha+\phi_n(\pi)$	
2	ω	0	ϕ_n+1
3	$[0,0,\gamma+1]$	$[0,0,\gamma]$	$\phi_n+[0,0,\gamma]$
4	$[\alpha+1,0,0]$	$[\alpha,0,0]$	$[\alpha,\phi_n,\phi_n]$
5	$[\lambda,0,0]$	$[\phi_n(\lambda),0,0]$	
6	$[\alpha,\beta+1,0]$	$[\alpha,\beta,0]$	$[\alpha,\beta,\phi_n]$
7	$[\alpha,\lambda,0]$	λ	$[\alpha,\phi_n(\lambda),\phi_n]$
8	$[\alpha+1,0,\gamma+1]$	$[\alpha+1,0,\gamma]$	$[\alpha,\phi_n,\phi_n]$
9	$[\lambda,0,\gamma+1]$	$[\lambda,0,\gamma]$	$[\phi_n(\lambda),\phi_n,\phi_n]$
10	$[\alpha,\beta+1,\gamma+1]$	$[\alpha,\beta+1,\gamma]$	$[\alpha,\beta,\phi_n]$
11	$[\alpha,\lambda,\gamma+1]$	$[\alpha,\lambda,\gamma]$	$[\alpha,\phi_n(\lambda),\phi_n]$
12a	$[\alpha,\beta,\lambda]$	$[\alpha,\beta,\phi_n(\lambda)]$	
12b1	$[0,0,\lambda]$	λ	$\phi_n+\lambda$
12b2	$[\alpha+1,0,\lambda]$	λ	$[\alpha,\phi_n,\phi_n]$
12b3	$[\mu,0,\lambda]$	λ	$[\phi_n(\mu),\phi_n,\phi_n]$
12b4	$[\alpha,\beta+1,\lambda]$	λ	$[\alpha,\beta,\phi_n]$
12b5	$[\alpha,\mu,\lambda]$	λ	$[\alpha,\phi_n(\mu),\phi_n]$

In a first stage, we define a function for each row of the table. Write $f'(x)$ instead of $\phi_n(\lambda)$ and $f(x)$ instead of $\phi_n(x)$. Each line has the form $f(n+1) = \mathrm{F}(x,f(n),f'(n))$. It may happen that there is no f' in F. Here we define f by induction on n and prove some properties. It may happen that there is no f in F. Here we define f as a normal function and prove some properties, assuming that f' satisfies some properties. The general case is similar.

In any case, we prove $\mathrm{L}(f,x)$, namely that f is strictly increasing and that x is the supremum of f. Note that f is strictly increasing if $f(n) < f(n+1)$; this is in general obvious. The property $f(n) < x$ is also generally easy. Consider $f(n) = [0,0,1,n,0]$. If $b > 1$ then $[0,0,b]$ is an upper bound for f. This function has no supremum (there is no least upper bound). For this reason, we consider only upper bounds that are NF; in the previous case $b > 1$ gives $b \geq 2$, and the supremum is $[0,0,2]$. To say that x is a supremum means: if b is an upper bound, then $x \leq b$. This is *not* the good definition. We shall use instead: if $t < x$, then there exists n such that $t \leq f(n)$. Assume $t = [a,b,c,n,d]$ and $x = [a',b',c']$. Then $t < x$ is the same

as $[a,b,c] < [a',b',c']$. We shall write this as (H). This condition may imply $b < x$ or $c < x$. For this reason, we proceed by induction on the size of t. If we meet $b < x$, we can replace it by $b \leq f(\mathrm{N})$ for some N.

```
Notation Tf := (nat -> T3).

Definition limit_of (f: Tf) x :=
  [/\ (forall n m, (n < m)%N -> f n < f m),
      (forall n, f n < x) &
      (forall y, T3nf y -> y < x -> (exists n, y <= f n))].

Definition limit12_hyp a b c:=
   if c is cons a1 b1 c1 n1 d1 then
       (n1 == 0) && (d1 == zero) &&
       ( ((a == a1) && (b < b1)) || ((a < a1) && (b < c)))
   else false.


Lemma fincP (f: Tf) :
  (forall n, f n < f n.+1) ->
  (forall n m, (n < m)%N -> f n < f m).
```

Consider case 1. Let's assume $\mathrm{L}(f,x)$. Define $g(n) = a+n$. Then $\mathrm{L}(g, a+x)$ holds, provided that a is NF.

Consider case 5. Let's assume $\mathrm{L}(f,x)$. Define $g(n) = [f(n),0,0]$. Then $\mathrm{L}(g,[x,0,0])$. Here (H) simplifies to $a < x$, $b < [x,0,0]$ and $c < [x,0,0]$. By assumption there is 1 such that $a \leq f(n_1)$; by induction there is n_2 such that $b \leq g(n_2)$, and n_3 such that $c \leq g(n_3)$. If m is greater then these three integer we get $t \leq g(m)$.

Consider case12a. Let's assume $\mathrm{L}(f,z)$ and $\mathrm{H}_b(x,y,z)$ is false. We define $g(n) = [x,y,f(n)]$ and pretend $\mathrm{L}(g,[x,y,z])$. Assume $t = [a,b,c,n,d] < [x,y,z]$. We have nine cases to consider, G6a to G6i. In case G6a we have $c < z$, and we conclude via $\mathrm{L}(f,z)$. In case G6b, $c < [x,y,z]$, and we proceed by induction. In cases G6c and G6h, we have $[a,b,c] < z$, and we conclude as in case G6a. In case G6d we have $[a,b,c] = z$. This contradicts H_b. In case G6e, we have $b < [x,y,z]$, and $c < [x,y,z]$; same as G6b. In cases G6f and G6g, we have $t \leq g(1)$. In case G6i we have $[a,b,c] = z$, and (H_b) says $y \leq z$, thus $t \leq g(1)$.

```
Definition phi0:= fun _ :nat => zero.
Definition phi1 a (f:Tf) := fun n => a + f n.
Definition phi5 (f:Tf) := fun n => [f n, zero,zero].
Definition phi12a a b (f:Tf) := fun n => [a,b,f n].

Lemma limit1 a b f:
   T3nf a -> (limit_of f b) ->  (limit_of (phi1 a f) (a + b)).
Lemma limit5 f x: (limit_of f x) -> (limit_of (phi5 f) [x,zero,zero]).
Lemma limit12a f a b c:   ~~ (limit12_hyp a b c) ->
   (limit_of f c) ->  (limit_of (phi12a a b f)[a, b, c]).
```

Let $\Phi_3(n) = f$ be the function $n \mapsto x \cdot (n+1)$ (defined by $f(n+1) = f(n) + x$). We have $\mathrm{L}(f,\mathrm{N}(x))$. We can restate this as: if $x = [0,0,a]$ then $\mathrm{L}(f,[0,0,a+1])$ (In particular, if $a = 0$, thus $x = 1$, we get $\mathrm{L}(f,\omega)$). On the other hand, if $\mathrm{H}_b(0,0,x)$ holds, then x has the form $x = [a_1,b_1,c_1]$ where one of a_1, b_1 is non-zero. We have then $\mathrm{L}(f,[0,0,x])$. In any case, if $t = [a,b,c,n,d]$ is less than the supremum, we have $t \leq f(n+1)$.

Let $\Phi_4(x)$ be the function defined by $f(0) = [x,0,0]$ and $f(n+1) = [x, f(n), f(n)]$. We have $\mathrm{L}(f,[x+1,0,0]))$. Here (H) is $a \le x$, $b < [x+1,0,0]$, $c < [x+1,0,0]$. By induction, there is m such that $b \le f(m)$, and there is m' such that $c \le f(m')$. We have $t \le f(\sup(m,m')+2)$.

Let $\Phi_8(x,y)$ be the function defined by $f(0) = [x+1,0,y]$ and $f(n+1) = [x, f(n), f(n)]$. We have $\mathrm{L}(f,[x+1,0,y+1]))$. Here (H) is a bit complicated; assume first $a < x+1$. We get $b < [x+1,0,y+1]$ and $c < [x+1,0,y+1]$. In this case, we proceed as for ϕ_4. Assume $x+1 < a$. The condition becomes $[a,b,c] \le y+1$. Since a is non-zero, this is $[a,b,c] \le y$. It follows $t \le f(0)$. Assume $a = x+1$. If b is non-zero, we get $[a,b,c] \le y+1$, and we conclude as before. If $b = 0$, we get $c < y+1$. If $c < y$ we have $t \le f(0)$. Otherwise $c = y$ and $t \le f(1)$.

Let $\Phi_{12b2}(x,y)$ be the function defined by $f(0) = y$ and $f(n+1) = [x, f(n), f(n)]$. We have $\mathrm{L}(f,[x+1,0,y]))$. Note that ϕ_4 and ϕ_8 are particular cases of this function. We shall assume here $\mathrm{H}_b(x+1,0,y)$. The proof is exactly the same as for ϕ_8, except for the last case, where H_b applies.

Let $\Phi_6(x,y)$ be the function defined by $f(0) = [x,y,0]$ and $f(n+1) = [x,y,f(n)]$. We have $\mathrm{L}(f,[x,y+1,0]))$. Condition (H) says that either $c < [x,y+1,0]$ or $[a,b,c] \le y+1$. In the first case we proceed by induction. In the second case, case to distinguish the subcases $a = 0$, and a non-zero, where the condition becomes $[a,b,c] < y$.

Let $\Phi_{10}(x,y)$ be the function defined by $f(0) = [x,y+1,z]$ and $f(n+1) = [x,y,f(n)]$. We have $\mathrm{L}(f,[x,y+1,z+1]))$. Same as above.

Let $\Phi_{12b4}(x,y,z)$ be the function defined by $f(0) = z$ and $f(n+1) = [x,y,f(n)]$. We have $\mathrm{L}(f,[x,y+1,z+1]))$. This is the same function as above, with a different initial value. We assume $\mathrm{H}_b(,y+1,z)$. Same proof as above.

```
Fixpoint phi3 x n := if n is n.+1 then phi3 x n + x else x.
Fixpoint phi4 x n :=
   if n is n.+1 then [x, phi4 x n, phi4 x n] else [x,zero,zero].
Fixpoint phi8 x y n :=
   if n is n.+1 then [x, phi8 x y n, phi8 x y n] else [T3succ x,zero,y].
Fixpoint phi12b2 x y n :=
   if n is n.+1 then [x, phi12b2 x y n, phi12b2 x y n] else y.
Fixpoint phi6 x y n :=
   if n is n.+1 then [x, y, phi6 x y n] else [x,y,zero].
Fixpoint phi10 x y z n :=
   if n is n.+1 then [x, y, phi10 x y z n] else [x,T3succ y,z].

Lemma phi3v a b c k: phi3 [a,b,c] k = cons a b c k zero.
Lemma limit3 x: limit_of (phi3 [zero,zero,x]) [zero, zero, T3succ x].
Lemma limit2: limit_of T3nat omega.
Lemma limit12b1  x:  (limit12_hyp zero zero x) ->
  limit_of (phi3 x) [zero, zero, x].
Lemma limit4 x: limit_of (phi4 x) [T3succ x, zero, zero].
Lemma limit8 x y: limit_of (phi8 x y) [T3succ x, zero, T3succ y].
Lemma limit12b2 x y: (limit12_hyp (T3succ x) zero y) ->
    limit_of (phi12b2 x y) [T3succ x, zero, y].
Lemma limit6 x y: limit_of (phi6 x y) [x,T3succ y, zero].
Lemma limit10 x y z:  limit_of (phi10 x y z) [x,T3succ y, T3succ z].
Lemma limit12b4 x y z: (limit12_hyp x (T3succ y) z) ->
    limit_of (phi12b4 x y z) [x,T3succ y,z].
```

We consider now cases where one argument y is limit and $\mathrm{L}(f',y)$ holds. Our function f will depend of f'.

Consider $\Phi_7(x,y,f')$ the function such $f(0)=y$ and $f(n+1)=[x,f'(n),f(n)]$. Assume $L(f',y)$. Then $L(f,[x,y,0])$. Same method as above.

Consider $\Phi_9(x,y,f')$ the function such $f(0)=[x,0,y]$ and $f(n+1)=[f'(n),f(n),f(n)]$. Assume $L(f',x)$. Then $L(f,[x,0,y+1])$. Same method as above.

Consider $\Phi_{11}(x,y,z,f')$ the function such $f(0)=[x,y,z]$ and $f(n+1)=[x,f'(n),f(n))]$. Assume $L(f',y)$. Then $L(f,[x,y,z+1])$. This case is detailed in [1].

Consider $\Phi_{12b3}(y,f')$ the function such $f(0)=y$ and $f(n+1)=[f'(n),f(n),f(n)]$. Assume $L(f',x)$ and $H_b(x,0,y)$. (this says $y=[a_1,b_1,c_1]$ where $x=a_1$ and $b_1\neq 0$, or $x<a_1$). Then $L(f,[x,0,y])$. There are four subcases for (H). In the first case we have H_b.

Consider finally $\Phi_{12b5}(x,z,f')$ the function such $f(0)=z$ and $f(n+1)=[x,f'(n),f(n)]$. Assume $L(f',y)$ and $H_b(x,y,z)$. Then $L(f,[x,y,z])$.

```
Fixpoint phi7 x y f n :=
  if n is n.+1 then [x, f n, phi7 x y f n] else y.
Fixpoint phi9 x y f n :=
  if n is n.+1 then [f n, phi9 x y f n,  phi9 x y f n] else [x, zero,y].
Fixpoint phi11 x y z f n :=
  if n is n.+1 then [x,f n, phi11 x y z f n ] else [x, y,z].
Fixpoint phi12b3 y f n :=
  if n is n.+1 then [f n, phi12b3 y f n , phi12b3 y f n] else y.
Fixpoint phi12b5 x z f n :=
  if n is n.+1 then [x,f n, phi12b5 x z f n ] else z.

Lemma limit7 x y f: (limit_of f y)  ->
  (limit_of ( phi7 x y f) [x,y,zero]).
Lemma limit9 x y f: T3nf y -> (limit_of f x)  ->
  (limit_of (phi9 x y f) [x,zero, T3succ y]).
Lemma limit11 x y z f: (limit_of f y)  ->
  (limit_of (phi11 x y z f) [x, y, T3succ z]).
Lemma limit12b3 x y f: (limit_of f x)  -> (limit12_hyp x zero y) ->
  (limit_of (phi12b3 y f) [x, zero, y]).
Lemma limit12b5 x y z f: (limit_of f y)  -> (limit12_hyp x y z) ->
  (limit_of (phi12b5 x z f) [x,y,z]).
```

We define our function ϕ is the same way as comparison. This means that we consider some formula F, define $\phi_{k+1}(x)=F(x,\phi_k)$, then $\phi(x)=\phi_{l(k)+1}(x)$. The objective is then to show $\phi(x)=F(x,\phi)$. The first argument to F is any element of T_3, and we are interested in the case where x is NF and limit; the second argument is a function f, such $f(y)$, as well as $F(x,f(y))$ are functions $\mathbf{N}\to T_3$.

We first define F in the case $x=[a,b,c]$; each argument is zero, a successor or a limit. If x is a successor, then it is the successor of its predecessor. Assume first $c=0$. We look at b. If $b=0$, we look at a. If $a=0$, we get $x=1$ which is not limit. If $a=a'+1$, we take $\Phi_4(a')$, and if a is limit, we take $\Phi_5(f(a))$. If $b=b'+1$, we take $\Phi_6(a,b')$, and if b is limit, we take $\Phi_7(a,b,f(b))$. Assume now $c=c'+1$. If $b=0$, we look at a. If $a=0$, we consider $\Phi_3(c')$, if $a=a'+1$, we consider $\Phi_8(a',c')$, and if a is limit, we consider $\Phi_9(a,c',f(a))$. Assume $b=b'+1$, we consider $\Phi_{10}(a,b',c')$, otherwise $\Phi_{11}(a,b,c',f(b))$. Consider finally the case where c is limit. If $H_b(a,b,c)$ is false, we consider $\Phi_{12a}(a,b,f(c))$. Otherwise, we look at b. If $b=0$, then if $a=0$ we take $\Phi_3(c)$, if $a=a'+1$, we take $\Phi_{12b2}(a',c)$, otherwise $\Phi_{12b3}(c,f(a))$. If $b=b'+1$ we take $\Phi_{12b4}(a,b',c)$, otherwise $\Phi_{12b5}(a,c,f(b))$.

Now $\phi(x)$ is defined as follows. If $x=0$, we take Φ_0. Assume $x=[a,b,c,n,d]$. If d is non-zero, we use $\Phi_1(x',f(d))$, where $x'=[a,b,c,n,0]$. If $d=0$, $n=m+1$, $x'=[a,b,c,m,0]$ we use

$\Phi_1(x', f([a,b,c])$, where $f([a,b,c])$ is the formula defined above.

```
Definition phi_rec_psi f a b c :=
  if (c==zero) then
    if(b==zero) then
       if(a==zero) then phi0
       else if(T3is_succ a) then phi4 (T3pred a)
       else phi5 (f a)
    else if(T3is_succ b) then phi6 a (T3pred b)
    else phi7 a b (f b)
  else if(T3is_succ c) then
    if(b==zero) then
      if(a==zero) then phi3 [zero,zero, T3pred c]
      else if (T3is_succ a) then phi8 (T3pred a) (T3pred c)
      else phi9 a (T3pred c) (f a)
    else if(T3is_succ b) then phi10 a (T3pred b) (T3pred c)
    else phi11 a b (T3pred c) (f b)
  else if (limit12_hyp a b c) then
    if(b==zero) then
      if(a==zero) then phi3 c
      else if(T3is_succ a) then phi12b2 (T3pred a) c
      else phi12b3 c (f a)
    else if (T3is_succ b) then phi12b4 a (T3pred b) c
    else phi12b5 a c (f b)
  else phi12a a b (f c).
Definition phi_rec f x :=
  if x is cons a b c n d then
  if (d==zero) then
    if n is n.+1 then phi1 (cons a b c n zero) (phi_rec_psi f a b c)
     else phi_rec_psi f a b c
  else phi1 (cons a b c n zero) (f d)
  else phi0.

Fixpoint phia k := if k is k.+1 then phi_rec (phia k) else (fun x =>phi0).
Definition phi x := phia (size x).+1 x.
```

That ϕ satisfies the equation is obvious. Let $x_n = [a,b,c,n,0]$. We have $\phi(x_{n+1}) = \Phi_1(x_n, x_0)$, where $x_0 = [a,b,c]$.

```
Lemma phiE x : phi x = phi_rec phi x.
Lemma phiE_1 a b c n:
  phi (cons a b c n.+1 zero) = phi1 (cons a b c n zero) (phi [a, b, c]).
Lemma phiE_2 a b c n d: d != zero ->
  phi (cons a b c n d) = phi1 (cons a b c n zero) (phi d).
Lemma phiE_3 a b c:  phi ([a,b,c]) = phi_rec_psi phi a b c.
```

Let's show that if x is NF and limit, then $L(\phi(x), x)$. As usual, the proof is by induction on the size of x. Assume $x = [a,b,c,n,d]$, write $x_n = [a,b,c,n,0]$ and $y = [a,b,c]$. By induction, the property holds for d, as well as for y (by careful analysis of the definition of ϕ, and all the previous lemmas). If $n = 0$ and $d = 0$, we have $x = y$, and the result is clear, or $d \neq 0$, case where $x = x_n + d$, or $n \neq 0$, case where $x = x_{n_1} + y$; in these two cases we apply `limit1`.

```
Theorem phiL x: T3nf x -> T3limit x -> limit_of (phi x) x.
```

4.6 Interpretation

Ackermann says that $(1,1,1)$ is ω, $(1,2,1)$ is the first transfinite epsilon, if $x < \epsilon_1$ then $(1,1,\alpha) = \omega^\alpha$. With our notations, we have $[0,0,1] = \omega$ and $[0,1,0] = \epsilon_0$. We have $[0,0,\alpha] = \omega^\alpha$ whenever $x < \epsilon_0$ (this the first transfinite epsilon). Ackermann considers $(1,1,(1,2,1))$, i.e. $[0,0,\epsilon_0]$. This is the limit of $\Phi_3(\epsilon_0)$, thus is $\epsilon_0 \cdot \omega = \omega^{\epsilon_0+1}$. This means that $[0,0,x]$ is not always ω^x. Ackermann says: let X = $[0,2,0]$; this is the least solution to $x = \epsilon_x$. If $\alpha <$ X, then $[0,1,\alpha] = \epsilon_\alpha$.

Let A be the set of NF ordinals x satisfying: for all y, if $y < x$, then $y + x = x$. Let f be the enumeration of A. This means that f is strictly increasing and its image is A. We define ω^x to be $f(x)$. Let B the set of all NF x such that $f(x) = x$, and g the enumeration of B. We define ϵ_x to be $g(x)$. In what follows, we describe the sets A and B and the functions f and g.

```
Lemma conc1 (x:= [zero,zero, epsilon0]): limit_of (phi3 epsilon0) x.
```

Let A$'$ be the set ordinals x satisfying: for all y, if $y < x$, then $y + x = x$. Note that zero and one satisfy this condition, but we shall exclude zero from A$'$. The condition is equivalent to x has the form $[a,b,c]$ and to: for all u, v, if $u < x$ and $v < x$, then $u + v < x$. Let A be the subset of A$'$ formed of the NF ordinals. If x is NF, and if for all NF ordinals u, v, the conditions $u < x$ and $v < x$ imply $u + v < x$, then $x \in$ A.

```
Definition T3ap x :=
  if x is cons a b c n d then ((n==0) && (d==zero)) else false.
Lemma ap_limit x: T3ap x -> (x == one) || (T3limit x).
Lemma ap_pr0 a b c (x := [a,b,c]) u v:
  u < x -> v < x -> u + v < x.
Lemma ap_pr1 c:
   (forall a b,  a < c -> b < c -> a + b < c)  ->
   (c== zero) || T3ap c.
Lemma ap_pr2 c:
   T3nf c -> c <> zero ->
   (forall a b, T3nf a -> T3nf b ->  a < c -> b < c -> a + b < c)  ->
   T3ap c.
Lemma ap_pr3 a b c y (x := [a,b,c]): y < x -> y + x = x.
Lemma ap_pr4 x: (forall b, b < x -> b + x = x) -> (x == zero) || T3ap x.
Lemma ap_pr2CE (c := cons zero zero T3bad 1 zero):
   (forall a b, T3nf a -> T3nf b ->  a < c -> b < c -> a + b < c).
```

Let f be the enumeration of A. This means that $f(x)$ is strictly increasing, and its image is A. Let $x = f(a)$ and $y = f(a+1)$. There is no element of A between x and y. We restate this as: y is the A-*successor* of x. Assume a limit. Consider all $f(b)$ for $b < a$. The supremum of all these $f(b)$ is in A, thus is of the form $f(t)$, and clearly $t = a$. Finally, A has a least element, which is $f(0)$. There are some non-trivial points here: (1) show that the set of all $f(b)$ has a supremum, (2) show that it is in A, and (3) show that these conditions allow us to define a function f. Note that if (1) fails, one can prove instead (4) there is some limit ordinal a, such that f is defined for $b < a$, and such that the sequence $f(b)$ (for $b < a$) is unbounded. In both cases, f is called the *enumeration* of A. If A satisfies condition (2) , we say that A is *closed*.

We study here the notion of A-successor. Let's define N(x) as follows: If $x = [a,b,c]$ and $a = b = 0$, then N(x) = $[0,0,c+1]$ otherwise N(x) = $[0,0,x]$. If $x \in$ A$'$ then $x <$ N(x). Conversely, assume $x = [a,b,c]$, $y = [a',b',c']$ and $x < y$; then N(x) $\leq y$ provided that either (a) one of a and b are non-zero or (b) c' is NF. This show that N(x) is the A-successor of x. Moreover,

in case (a), it is also the A′-successor of x. However, consider $x = [0,0,1]$, $y = [0,0,z]$; the condition $x < y$ is equivalent to $1 < z$; there is no least z satisfying this property (the least NF z is of course 2).

```
Definition psi_succ x :=
  if x is cons a b c _ _ then
   if ((a==zero) && (b==zero)) then [zero,zero, T3succ c] else [zero,zero, x]
  else zero.

Lemma psi_succ_pr1 a b c: [a,b,c] < psi_succ ([a,b,c]).
Lemma succ_psi1 a b c (x:= [a, b, c]): ((a != zero) || (b != zero)) ->
    (forall a' b' c',  x < [a',b',c'] -> (psi_succ x) <=  [a',b',c']).
Lemma succ_psi2 u (x := [zero,zero,u]) :
   (forall a' b' c', T3nf c' -> x < [a',b',c'] -> (psi_succ x) <=  [a',b',c']).

Lemma succ_prCE  (u:= one) (v := T3bad):  (u < v) && (v < T3succ u).
Lemma succ_psiCE (z := [zero,zero, T3bad]):
   (omega < z) && (z < (psi_succ omega)) && ~~(T3nf z).
```

Let f be a function $T_3 \to T_3$, $x \in T_3$. We say that z is the supremum of $f(y)$, $y < x$, if z is an upper bound, and if $z' < z$, then there is $y < x$ such that $z' < f(y)$. We say that f is normal if f is strictly increasing and x is the supremum of $f(y)$, $y < x$, whenever x is limit.

We show here that the supremum is unique, compute the supremum when f is the identity function, show that the identity is normal, addition is normal and that composition of normal functions is normal.

```
Definition sup_of (f: T3-> T3) x z :=
  [/\ T3nf z,
      (forall y, T3nf y -> y < x -> f y <= z) &
      (forall z', T3nf z' -> z' < z -> exists y,
          [&& T3nf y, y < x & z' < f y])].
Definition normal f:=
  [/\ forall x, T3nf x -> T3nf (f x),
      (forall x y, T3nf x -> T3nf y -> x < y -> f x < f y)&
      (forall x, T3nf x -> T3limit x -> sup_of f x (f x)) ].

Lemma sup_unique f x z z': sup_of f x z -> sup_of f x z' -> z = z'.
Lemma sup_0alpha_zero: sup_of id zero zero.
Lemma sup_0alpha_limit x: T3nf x -> T3limit x -> sup_of id x x.
Lemma sup_0alpha_succ x: T3nf x -> sup_of id (T3succ x) x.
Lemma normal_id: normal id.
Lemma normal_limit f x: normal f -> T3nf x -> T3limit x -> T3limit (f x).
Lemma normal_compose f g:
   normal f -> normal g -> normal (f \o g).
Lemma add_normal a:  T3nf a -> normal (T3add a).
```

Chapter 5

Link with von Neumann ordinals

From now on, the term "ordinal" will design a von Neumann ordinal, as defined in [6]. One can consider an ordinal as a set satisfying some properties (for instance $\forall x, x \notin 0_o$ defines 0_o to be the empty set, $\forall x, x \in y^+ \iff (x \in y \vee x = y)$ defines the successor y^+ of y). We give here some operational rules: 0_o is an ordinal, if y is an ordinal, then y^+ is an ordinal, ordinals can be compared via $\leq_o$, and this comparison is a well-ordering (in particular, one can prove properties by transfinite induction). The important point is that one can define functions by transfinite induction, for instance addition $a +_o b$, multiplication $a \cdot_o b$, exponentiation, and so one.

Some functions are "normal" (for instance, addition is normal when the first argument is fixed). A normal function f has many fixed points (ordinals x such that $f(x) = x$). This means that there exists a strictly increasing function, g such $f(x) = x$ is equivalent to: there is y such that $x = g(y)$. The function g is called the first derivation of f, and is normal, so that one can consider the second, third, etc, derivation of f.

Take for instance $f(x) = 1 + x$. Let ω_0 be the least fixed point of f; then all ordinals $\geq \omega_0$ are fixed-points of f. Thus, the first derivation of f is $x \mapsto \omega_0 + x$. The set of ordinals $< \omega_0$ is naturally isomorphic to the set of natural numbers. Some ordinals are called "cardinals", and ω_i denotes the enumeration of cardinals $\geq \omega_0$. All ordinals considered here are "countable", i.e., in bijection with ω_0, or in other terms, $< \omega_1$. For simplicity, we shall write ω instead ω_0.

Take $f(x) = \omega^x$. An ordinal of the form $f(x)$ is called AP (additive principal), any ordinal x can uniquely be written as a sum of AP numbers, $x_1 + x_2 + \ldots + x_k$, where the sequence is decreasing. Note that $k = 0$ if and only if $x = 0$. Assume x non-zero, let $\mathrm{C}(x)$ be the first term x_1, and $n+1$ the number of occurrences of x_1 in the sum. We can write

$$x = \mathrm{C}(x) \cdot (n+1) + \mathrm{R}(x) \tag{5.1}$$

and (by definition of n) we have: $\mathrm{R}(x) < \mathrm{C}(x)$.

Let ϵ_x be the first derived function of $f(x) = \omega^x$. In particular, ϵ_0 is the least ordinal x such that $\omega^x = x$. This implies: if $x < \epsilon_0$ then $\mathrm{C}(x)$ and $\mathrm{R}(x)$ are $< \epsilon_0$.

Let $f(x, n)$ be the n-th derivation of f applied to x. Veblen explains how to define this function for arbitrary ordinals n, instead of mere integers; if $f(x)$ is ω^x, the result is denoted $\phi(a, b)$; one may deduce a function $\psi(a, b)$; these functions satisfy equations (3.1) and (3.2). Every ω^x is uniquely of the form $\psi(a, b)$, so that (5.1) can be rewritten as

$$x = \psi(x_1, x_2) \cdot (n+1) + \mathrm{R}(x) \tag{5.2}$$

The least x with $\phi(x,0) = x$ is denoted Γ_0, and called the Feferman-Schütte ordinal. If $x < \Gamma_0$, the quantities x_1, x_2 and $R(x)$ defined by (5.2) are all $< \Gamma_0$.

One can generalize the Veblen construction: from $f(x)$ we get $f(x,y)$, then $f(x,y,z)$, etc. [This is not yet implemented in COQ]. Veblen considers a function $f(x_1, x_2, \ldots)$ with an arbitrary number of arguments; this means that the argument of f is a function x, such that for every ordinal i, x_i is an ordinal; the value of x_i is zero except for a finite number of ordinals i. The equivalent of Γ_0 is called the small Veblen ordinal (if the number of arguments is finite), or the large Veblen ordinal (if the number of arguments is infinite), sometimes denoted $\phi_{\Omega^\omega}(0)$ or $\phi_{\Omega^\Omega}(0)$; here $\Omega = \omega_1$ is the set of all countable ordinals, and an element of Ω^ω (resp. Ω^Ω) is a finite (resp infinite countable) sequence of countable ordinals.

Let T be one of the three type T_1, T_2, T_3 described in the previous chapters. Let T' be the NF elements of T. We have a relation $\leq$, which is a well-founded total order relation, thus a well-ordering. Consider T' as a set E in the Bourbaki sense. The comparison on T' induces a comparison on E that makes it a well-ordered set; there is a unique function $f : E \to O(\alpha)$ such that α is an ordinal, $O(\alpha)$ is the set of ordinals less than α, and f is an order-isomorphism. This means: there is a unique ordinal α and function f, such that f maps NF elements of type T to an ordinal $< \alpha$, and f is an order isomorphism.

In this chapter, we study the function f. This is not so easy. We first postulate that f satisfies the following property. Every x of T can be written as $x = x_0 \cdot (n+1) + r$, where n is an integer, $r < x_0$ and x is AP (Recall that an element x of T has the form $[a,n,r]$, $[a,b,n,r]$ or $[a,b,c,n,r]$; here x_0 is obtained from x by replacing n and r by zero). We assume $f(x)+f(y) = f(x+y)$ and that f maps an AP element to an AP ordinal [we believe that this postulate can be proved]. Write relation (5.1) for $y = f(x)$ as $y = C(y) \cdot (n+1) + R(y)$, so that $f(x_0) = C(y)$, $f(r) = R(y)$. This says that f is uniquely defined by the values $f(x_0)$. We shall also postulate that $f(x_0)$ has the form $g(f(a))$, $G(f(a), f(b))$ or $\Psi(f(a), f(b), f(c))$ in the cases T_1, T_2, T_3. [This is not completely obvious]. This means that we introduce a function g, G or Ψ, and study the properties. The assumptions are: the value of the function is an AP, and $g(x) < g(y)$ or $\Psi(a,bc) < \Psi(a',b',c')$ is equivalent to the ordering of T_1 or T_3; this relation is simple in the first case, complex in the last one. At the end of the previous chapter we have mentioned $\Psi(0,0,a) = \omega^a$ for $a < \epsilon_0$.

One question is: what is the range of f?, equivalently, what is α. It is ϵ_0 in case T_1, Γ_0 in case T_2 and larger in case T_3. It is obviously less than the small Veblen ordinal but nothing else in known.

5.1 First approach

We consider here only the type T_1, but the generalization is obvious. We assume that there is an injective function Ro, denoted $\mathscr{R}$, such that, if $b = \mathscr{R}(a)$, then a of type T_1 says $b \in T_1$ (where b is a Bourbaki set, and $\in$ is the membership relation on Bourbaki sets; see [5] for details), Moreover, there is Bo, denoted $\mathscr{B}$, such that if H is the assumption $b \in T_1$, then $\mathscr{B}(H)$ is some a of type T_1; we have $\mathscr{R}(a) = b$. Note that, if a is of type T_1 and H is a proof of $\mathscr{R}(a) \in T_1$, then $\mathscr{B}(H) = a$. There are the properties we can use:

```
Lemma Ex1: forall a:T1, inc (Ro a) T1.
  Proof. by move => a; apply:R_inc. Qed.
Lemma Ex2: forall a b:T1, Ro a = Ro b -> a = b.
  Proof. by move => a b ; apply:R_inj. Qed.
Lemma Ex3: forall (b:Set)(Ha:inc b T1), Ro (Bo Ha) = b.
```

```
  Proof. by move => b; apply:B_eq. Qed.
Lemma Ex4: forall (a:T1)(Ha:inc (Ro a) T1), (Bo Ha) = a.
  Proof. by move => a; apply:B_back. Qed.
```

Let E be T_1 considered as a Bourbaki set. We want to equip E with an order relation compatible with the relation $\leq$ of T_1. We are looking for a subset G of $E \times E$ such that $(x, y) \in G$ is equivalent to $a \leq b$, where $x = \mathscr{R}(a)$ and $y = \mathscr{R}(b)$. Note that a is unique, it exists since G is a subset of $E \times E$. However we cannot use $G \subset E \times E$ in the definition of G. Assume that there is a function i such that $\mathscr{R}(i(x)) = x$ when $x \in E$. Then we can define G as the set of all $(x, y) \in E \times E$ such that $i(x) \leq i(y)$.

```
Definition  set_to_T1 x := match  (ixm (inc x T1) )  with
 | inl hx => Bo hx
 | inr _ => zero end.

Lemma set_to_T1_pr x (Hx: inc x T1) : set_to_T1 x = Bo Hx.
Lemma set_to_T1_inj x y: inc x T1 -> inc y T1 ->
   set_to_T1 x = set_to_T1 y -> x = y.
```

Consider now the relation $x \leq_1 y$: $x \in E$, $y \in E$ and $i(x) \leq i(y)$, and its graph G on E (this is the set of all pairs (x, y) in $E \times E$ such that $x \leq_1 y$). Obviously $(x, y) \in G$ is equivalent to $x \leq_1 y$, and G defines an ordering on E.

```
Definition ST1_le x y := [/\ inc x T1, inc y T1 & set_to_T1 x <=  set_to_T1 y].
Definition ST1_order := graph_on ST1_le T1.
Lemma ST1_osr: order_on ST1_order T1.
Lemma ST1_leP x y: gle ST1_order x y <-> ST1_le x y.
```

Let E' be the subset of E formed of all x such that $i(x)$ is NF. We consider the order induced by G on E'. The associated relation $x <_E y$ is: $x \in E$, and $y \in E'$ and $x \leq_1 y$. We prove here that it is an order on E', a total order, a well-order. Note that $\leq_E$ is total since $\leq$ is total. Now we use Corollary 1 of Proposition 6 of Bourbaki: A totally ordered set is well-ordered if and only if every decreasing sequence is stationary. If E were not well-ordered, we could construct a non-stationary decreasing sequence, extract a strictly decreasing sequence (x_k). Set $a_k = i(x_k)$. We have $a_{k+1} <_N a_k$, where $<_N$ is the relation proved to be well-founded on page 18. Such a sequence cannot exist.

```
Definition T1N := Zo T1 (fun z => T1nf (set_to_T1 z)).
Definition STN_order:= induced_order  ST1_order T1N.

Lemma STN_osr: order_on STN_order T1N.
Lemma STN_tor: total_order STN_order.
Lemma STN_wosr: worder_on STN_order T1N.
```

Let α be the ordinal of $<_E$. This is some f, an isomorphism between the set of ordinals $< \alpha$ ordered by ordinal comparison, and E' ordered by $<_E$. One could easily prove the following: f maps zero to zero (since zero is the least element in each case), f maps an integer n to $F(N(n))$, where $N(n)$ denotes the Bourbaki integer n considered as object of type `nat` and F is `T1nat` (proof by induction, as $n+1$ is the successor of n in one case and $F(N(n+1))$ is the successor of $F(N(n))$, in the other case. One can also show that $f(\omega)$ is `T1omega` (consider the least element not of the form $F(N(n))$. Proving more facts is difficult.

```
Definition STN_iso:= the_ordinal_iso STN_order.
Definition ord_T1:= ordinal STN_order.

Lemma STN_iso_pr:  order_isomorphism STN_iso (ordinal_o ord_T1) STN_order.
```

5.2 Second Approach

Define by induction on T_1 a function O_1

$$O_1(0) = 0, \qquad O_1([a,n,b]) = \omega^{O_1(a)} \cdot (n+1) + O_1(b).$$

By induction, we have $O_1(x) < \epsilon_0$. On the other hand every ordinal less than ϵ_0 has the form $O_1(x)$, proof by transfinite induction on von Neumann ordinals: if the property holds for all ordinals less than x, then (5.1) says that it holds also for x.

```
Fixpoint T1toB (x: T1) : Set :=
  if x is cons a n b then
     omega0 ^o (T1toB a) *o (nat_to_B n.+1) +o  (T1toB b)
  else \0o.

Lemma OS_succN n : ordinalp (nat_to_B n.+1).
Lemma OS_T1toB x : ordinalp (T1toB x).
Lemma T1toB_small x : T1toB x <o epsilon0.
Lemma T1toB_surjective x: x <o epsilon0 -> exists y, x = T1toB y.
```

We now show that O_1 is strictly increasing, thus injective, proof by `T1transfinite_induction`. In fact, we show that, for all z, $x < y \leq z$ implies $O_1(x) < O_1(y)$ (all quantities being NF). The cases where $x = 0$ or $y = 0$ are trivial; so assume $[a,n,b] < [a',n',b']$, and let's show $\omega^A(n+1) + B < \omega^{A'}(n'+1) + B'$ (here $A = O_1(a)$, etc). If $a = a'$ and $n = n'$, we get $b < b'$, and by induction $B < B'$; this is equivalent to $u + B < u + B'$, whatever u, whence the result. We have $B < \omega^A$ since x NF says $b < \phi_0(a)$ (note that we cannot use structural induction here, we really need transfinite induction). Thus $X < \omega^A(n+2)$. Assume $a = a'$, $n < n'$. We conclude by $\omega^A(n+2) \leq \omega^A(n'+1) \leq Y$. Finally, if $a < a'$ we have $A < A'$, thus $X < \omega^{A'} \leq Y$.

```
Lemma T1toB_mon1 x y: T1nf x -> T1nf y ->
   x < y -> T1toB x <o T1toB y.
Lemma T1toB_mon2 x y: T1nf x -> T1nf y ->
   (x < y <-> T1toB x <o T1toB y).
Lemma T1toB_injective x y: T1nf x -> T1nf y ->
  T1toB x = T1toB y -> x = y.
```

Consider now the type T_2 and

$$O_2(0) = 0, \qquad O_2([a,b,n,c]) = \psi(O_2(a), O_2(b)) \cdot (n+1) + O_2(c).$$

Relation (5.2) and properties of ψ say that the image of this function is the set of ordinals less than Γ_0. The non-trivial point is to check that it is an order isomorphism.

We show $x < y \implies O_2(x) < O_2(y)$ whenever x and y are NF, by induction on the size of the arguments. Assume $x = [a,b,n,c]$, and let $x' = [a,b]$. We have $O_2(x) = O_2(x') \cdot (n+1) + O_2(c)$. We have $c < [a,b]$ by normality. It follows $O_2(c) < O_2(x')$, but we cannot use the induction

assumption. However we have $c < x' \leq y'$; so that $O_2(c) < O_2(y')$ by induction. The conclusion follows in the case $x = x'$. We now have to show that $x' < y'$ implies $O_2(x') < O_2(y')$. Note that $O_2(x') = \psi(O_2(a), O_2(b))$, likewise for y'. The relation $\psi(O_2(a), O_2(b)) < \psi(O_2(c), O_2(d))$ is defined by (3.2). The first clause is "if $O_2(a) < O_2(c)$ then $O_2(b) < \psi(O_2(c), O_2(d))$". This is the same as: if $O_2(a) < O_2(c)$ then $O_2(b) < O_2(y')$, and is equivalent (by induction) to "if $a < c$ then $b < y'$". All three clauses can be handled similarly. So $O_2(x') < O_2(y')$ is equivalent to $[a, b] < [c, d]$.

```
Fixpoint T2toB (x: T2) : Set :=
  if x is cons a b n c then
    Schutte_psi (T2toB a) (T2toB b)  *o (nat_to_B n.+1) +o  (T2toB c)
  else \0o.

Lemma T2toB_small x : T2toB x <o Gamma_0.
Lemma OS_T2toB x : ordinalp (T2toB x).

Lemma T12oB_surjective x: x <o Gamma_0 -> exists y, x = T2toB y.
Lemma T2toB_mon1 x y: T2nf x -> T2nf y ->
   x < y -> T2toB x <o T2toB y.
Lemma T2toB_mon2 x y: T2nf x -> T2nf y ->
   (x < y <-> T2toB x <o T2toB y).
Lemma T2toB_injective x y: T2nf x -> T2nf y ->
 T2toB x = T2toB y -> x = y.
```

5.3 Third Approach

The idea here is the following: We have shown in the first chapter that we can define a function f on T' (the set of NF elements of T) so that $f(x)$ is $P(f_x)$ where f_x is the restriction of f to the set of elements less than x. Define P as follows: if $x = 0$, then the value is the ordinal zero. Assume that x is a successor. Then x is the successor of some y, and $y < x$. We know $f(y)$, so we can define $f(x)$ as the (von Neumann) successor of $f(y)$. Otherwise, we define $f(x)$ is the supremum of the $f(y)$ (for $y < x$).

What we get is a strictly increasing function that maps an element of T onto an ordinals, it maps a successor onto a successor and a limit ordinal onto a limit ordinal. This is perhaps the best way to study the type T_3.

Bibliography

[1] Wilhemn Ackermann. Konstruktiver aufbau eines abschnitts der zweiten cantorschen zahlenklasse. *Mathemathische Zeitschrift*, 53(5):403–413, 1951.

[2] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development*. Springer, 2004.

[3] Georg Cantor. *Contributions to the Founding of the Theory of Transfinite Numbers*. Dover Publications Inc, 1897. Trans. P. Jourdain, 1955.

[4] Georges Gonthier and Assia Mahboubi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA, 2008. http://hal.inria.fr/inria-00258384/en/.

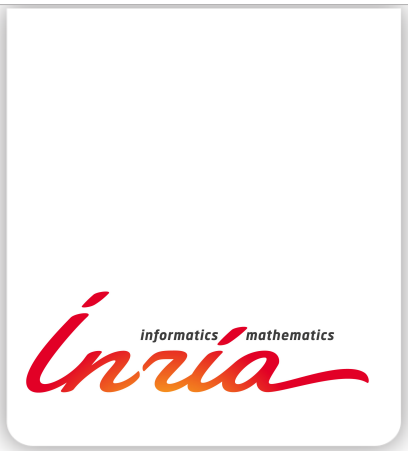
[5] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part One, Theory of Sets. Research Report RR-6999, INRIA, 2009. http://hal.inria.fr/inria-00408143/en/.

[6] José Grimm. Implementation of Bourbaki's Elements of Mathematics in Coq: Part Two; Ordered Sets, Cardinals, Integers. Research Report RR-7150, INRIA, 2009. http://hal.inria.fr/inria-00440786/en/.

[7] Kurt Schütte. *Proof theory*. Springer, 1977.

[8] Oswald Veblen. Continous increasing functions of finite and transfinite ordinals. *Transactions of ther AMS*, 9(3), 1908. DOI=10.2307/1988605.

Contents



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399